Titre: Title:	Méthodes pour favoriser l'intégralité de l'amélioration dans le simplexe en nombres entiers - Application aux rotations d'équipages aériens
Auteur: Author:	Samuel Rosat
Date:	2016
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Rosat, S. (2016). Méthodes pour favoriser l'intégralité de l'amélioration dans le simplexe en nombres entiers - Application aux rotations d'équipages aériens [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2072/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/2072/
Directeurs de recherche: Advisors:	François Soumis, & Issmail El Hallaoui
Programme: Program:	Doctorat en mathématiques de l'ingénieur

UNIVERSITÉ DE MONTRÉAL

MÉTHODES POUR FAVORISER L'INTÉGRALITÉ DE L'AMÉLIORATION DANS LE SIMPLEXE EN NOMBRES ENTIERS – APPLICATION AUX ROTATIONS D'ÉQUIPAGES AÉRIENS

SAMUEL ROSAT
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES DE L'INGÉNIEUR)
MARS 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

MÉTHODES POUR FAVORISER L'INTÉGRALITÉ DE L'AMÉLIORATION DANS LE SIMPLEXE EN NOMBRES ENTIERS – APPLICATION AUX ROTATIONS D'ÉQUIPAGES AÉRIENS

présentée par : ROSAT Samuel
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. GENDREAU Michel, Ph. D., président
M. SOUMIS François, Ph. D., membre et directeur de recherche
M. EL HALLAOUI Issmaïl, Ph. D., membre et codirecteur de recherche
M. ORBAN Dominique, Doctorat, membre
M. POGGI DE ARAGÃO Marcus Vinicius Soledade, Ph. D., membre externe

DÉDICACE

À Isabelle, Olivier et Lise

« Le véritable voyage commençait. Jusqu'alors les fatigues l'avaient emporté sur les difficultés ; maintenant celles-ci allaient véritablement naître sous nos pas. »

Jules Verne, Voyage au centre de la terre, chap. XVII.

REMERCIEMENTS

Il revient souvent que le doctorat est une aventure constituée de pics de stress, de réussites éphémères et de longues périodes de doute, mais que c'est avant tout un cheminement solitaire. Au moment d'écrire ces quelques lignes, il m'est pourtant difficile d'énumérer et de compter tous celles et ceux qui m'ont permis, chacun à sa façon, d'avancer dans mon périple tant vous êtes nombreux. Je tâcherai de n'oublier personne, et espère que ceux qui me connaissent et pensent l'avoir été y reconnaitront mon étourderie bien plus qu'une quelconque inimitié.

Il est des mots qui laissent des traces indélébiles. Pour cette raison, je souhaite remercier Thierry avant tout autre. Un jour de décembre 2012 ou 2013, je ne me rappelle plus la date exacte, tu m'as dis qu'il m'appartenait de continuer ou d'abandonner mon parcours doctoral, mais que choisir de continuer n'aurait de sens que si je menais le projet à terme. J'ai pris ma décision, et je m'y suis tenu, me remémorant notre discussion à de nombreuses reprises. Sans toi, je pense sincèrement que je n'aurais pas trouvé le courage d'aller au bout.

Ma gratitude ainsi que mes remerciements les plus chaleureux vont à mes directeurs de thèse, François et Issmail. Vous avez toujours étés disponibles et réactifs aux moments où j'en avais besoin. François, je te remercie pour ta patience, pour m'avoir écouté « japper » et pour m'avoir remotivé plus d'une fois. J'espère un jour savoir faire preuve d'autant de flegme que toi. Je te remercie pour avoir été plus qu'un directeur de recherche, mais aussi un partenaire de squash et de tarot avec qui j'ai toujours pris autant de plaisir à jouer et à échanger. Issmail, je te remercie pour être régulièrement venu frapper à la porte de mon bureau t'informer d'abord de moi et ensuite de l'avancement du projet, ainsi que pour toutes les discussions que nous avons eues. La seule difficulté consistait pour moi à faire tenir mes brouillons d'idées sur le (trop) petit tableau de ton bureau ! Merci aussi d'avoir lu et relu les détails de chacune des preuves, pas toujours claires et rarement tout à fait justes. Tous les deux, je vous remercie infiniement pour m'avoir toujours fait confiance.

Je remercie le jury, Michel Gendreau, Dominique Orban et Marcus Poggi pour avoir accepté de relire, critiquer et juger ce manuscrit.

Mes pensées se tournent ensuite vers Andrea. La tua esperienza sugli algoritmi primali e la tua connoscenza quasi illlimita di quel settore di ricerca hanno costituito, per me, un aiuto molto prezioso. Il tuo entusiasmo è profondamente contagioso e sono rimasto molto colpito del tuo benvenuto a braccia aperte quando sono arrivato a Bologna. Grazie anche per tutte le partite di calcetto.

Je remercie mes autres coauteurs, Driss et Frédéric, avec qui j'ai eu beaucoup de plaisir à collaborer.

Je remercie le CRSNG et KRONOS Inc. pour m'avoir financé durant toute la durée de cette thèse.

S'il ne reste que peu de typos, de fautes, de phrases trop lourdes ou alambiquées dans ce manuscrit, c'est grâce aux relectures attentives de Clément, François, Isabelle, Issmail et Thibault.

Je tiens aussi à remercier certains des professeurs que j'ai cotoyés au cours de ma scolarité car ils m'ont donné le goût de l'école, de l'apprentissage et de la recherche. Je pense en particulier à Monique, Mme Gaillet, M. Colin, Abou et Stéphane Gaubert.

Je remercie toute l'équipe du GERAD, particulièrement Francine pour tes réponses à mes questions sur LaTeX et pour avoir chaque fois fait des merveilles avec les fichiers source que je t'envoyais.

Qu'aurait été cette thèse sans mes compagnons de route Antoine, Jean-Bertrand, Jérémy, Marilène et Thibault ? Vous m'avez suivi de Montréal à Prague, de Barcelone à Paris et de Bologne à Pittsburgh. L'aventure fut bien plus excitante grâce à vous et je garderai un merveilleux souvenir des journées de travail et des nombreuses soirées autour d'un ou plusieurs verres. Je garde bizarrement moins de souvenirs des lendemains. Bref, vous êtes bien plus que des collègues, mais j'ai à peine besoin de vous le rappeler car vous le savez déjà. Merci aux joueurs de tarot qui m'ont souvent permis de décompresser le midi, Alain, Guy, Serge, Stéphane, et tous les autres. Je remercie aussi toute la *gang* du GERAD, Aïda, Alexis, Atoosa, Bastien, Charles, Claire, Filippo, Hélène-Sarah, Luca, Luca, Martin, Mélisende, Régis, Remy, Romain, Sara, Sébastien, Steve (pour ton acceuil au ZIB aussi), Vincent et tous les autres pour avoir apporté de la lumière jusque dans les bureaux sans fenêtres du GERAD.

J'ai été très touché par les bras qui m'ont étés ouverts quand je suis arrivé à Bologne et les liens que j'ai tissés au DEIS resteront. Claudio, Martin, Paolo, Sven e Tiziano, vi ringrazio col cuore per il vostro caloroso benvenuto a Bologna, grazie a voi non mi sono sentito un estraneo. Lavorare con voi nel corridorio della gloria è stato un grandissimo onore. Ringrazio anche Alberto, Andrea, Dimitri, Eduardo, Maxence e Valentina. Non mi dimenticherò mai né l'Osteria dei Griffoni né di Nonna Cesira, e sarò sempre orgoglioso di avere finito La Cotoletta del Mulino Bruciato sotto la supervisione dei più grandi maestri.

Loin de chez soi, on se sent parfois le besoin de reconstituer un cocon, un environnement familier et proche. Alexis, Audrey, Florian, Hélène, Loïc, Pierre, Nico, Sylvie et Thibault,

vous avez été, et êtes encore, cette nouvelle famille. Pour avoir cohabité avec moi des temps plus ou moins longs, mais toujours pour mon plus grand plaisir, je vous remercie infiniement. Merci du fond du cœur à Marie pour ton soutien et ta présence quotidienne pendant les deux premières années de cette thèse. Nos chemins se sont séparés mais je serais ô combien ingrat d'oublier tout ce que tu m'as apporté.

Merci à Lise et François d'avoir grandement facilité le processus de mon installation à Montréal. La richesse de l'environnement social dans lequel j'ai vécu ces années de thèse a assurément influencé mon efficacité et la qualité de mes recherches, bien que je ne sache dire dans quel sens. Je pense à ceux avec qui je suis arrivé ici et qui maintenant sont repartis, Benjamin, Casimir, Claire, Matthieu, Martin et Sarah, et à tous ceux que j'ai rencontrés, Clothilde, Jenna, Joris, Julie, Laura, Laurent, Myriam, Rémi, Samuel, et tant d'autres.

Le soutien qui m'a été apporté, à distance, par Anne-B., Claire, Clara, Daniel, Diane, Guillaume, Ji-Yun, Mustapha et Pierre m'a été très précieux. L'éloignement géographique a paradoxalement eu pour effet de resserrer nos liens et de me conforter dans l'idée que vous comptez tous beaucoup pour moi.

Manu, cela fait presque 25 ans que nous nous sommes assis pour la première fois, côte-à-côte sur un banc d'école. Je ne serais ce que je suis ni humainement ni académiquement sans la foultitude de débats mathématiques et métaphysiques que nous avons eus, de conneries que nous avons faites, et de délires que nous avons partagés. Tu as ta part de responsabilité dans ma décision de tenter l'aventure doctorale.

Enfin, je ne serais bien évidemment jamais arrivé jusqu'ici sans le soutien indéfectible de ma famille, à qui je dois tant. Je ne remercierai jamais assez mes parents, Isabelle et Olivier, qui m'ont donné la soif d'apprendre, la passion de transmettre et l'ambition de réussir. Lise, ma chère sœur, tu as toujours été et seras toujours ma plus grande *fan*, tout comme j'ai toujours été et serai toujours le tien. Tu es pour moi une source intarissable de motivation. Je pense aussi à Clément et Hugo, mes cousins, et à tout le reste de ma famille sans qui je n'aurais pu aller aussi loin.

RÉSUMÉ

Dans son cadre le plus général, le processus d'optimisation mathématique se scinde en trois grandes étapes. La première consiste à *modéliser* le problème, c'est-à-dire le représenter sous la forme d'un *programme mathématique*, ensemble d'équations constitué d'un objectif à minimiser ou maximiser (typiquement, les coûts ou le bénéfice de l'entreprise) et de contraintes à satisfaire (contraintes opérationnelles, convention collective, etc.). Aux décisions à prendre correspondent les variables du problème. S'il est une représentation parfaite de la réalité, ce modèle est dit exact, sinon il reste approximatif. La seconde étape du processus est la *résolution* de ce programme mathématique. Il s'agit de déterminer une solution respectant les contraintes et pour laquelle la valeur de l'objectif est la meilleure possible. Pour ce faire, on applique généralement un *algorithme de résolution*, ensemble de règles opératoires dont l'application permet de résoudre le problème énoncé au moyen d'un nombre fini d'opérations. Un algorithme peut être traduit grâce à un langage de programmation en un programme exécutable par un ordinateur. L'exécution d'un tel programme permet ainsi de résoudre le programme mathématique. Enfin, la dernière étape consiste à *ajuster* la solution obtenue à la réalité. Dans le cas où le modèle n'est qu'approximatif, cette solution peut ne pas convenir et nécessiter d'être modifiée a posteriori afin de s'accorder aux exigences de la réalité concrète. Cette thèse se concentre sur la seconde de ces trois étapes, l'étape de *résolution*, en particulier sur le développement d'un algorithme de résolution d'un programme mathématique précis, le *partitionnement d'ensemble*.

Le problème de partitionnement d'ensemble permet de modéliser des applications variées : planification d'emplois du temps, logistique, production d'électricité, partage équitable, reconnaissance de forme, etc. Pour chacun de ces exemples l'objectif et les contraintes prennent des significations physiques différentes, mais la structure du modèle est la même. D'un point de vue mathématique, il s'agit d'un programme linéaire en nombres entiers, dont les variables sont binaires, c'est-à-dire qu'elles ne peuvent prendre que les valeurs 0 et 1. Le programme est linéaire car l'objectif et les contraintes sont représentés par des fonctions linéaires des variables. Les algorithmes les plus couramment utilisés pour la résolution de tels problèmes sont basés sur le principe de séparation et évaluation (*branch-and-bound*). Dans ces méthodes, les contraintes d'intégralité sont d'abord relâchées : les solutions peuvent alors être fractionnaires. La résolution du programme ainsi obtenu – appelé *relaxation linéaire* du programme en nombres entiers – est bien plus simple que celle du programme en nombres entiers. Pour obtenir l'intégralité, on sépare le problème afin d'éliminer les solutions fractionnaires. Ces séparations donnent naissance à un arbre de branchement où, à chaque nœud, la

relaxation d'un problème de partitionnement de la taille du problème original est résolue. La taille de cet arbre, et donc le temps d'exécution, croissent exponentiellement avec la taille des instances. De plus, l'algorithme utilisé pour résoudre la relaxation, le *simplexe*, fonctionne mal sur des problèmes dégénérés, c'est-à-dire dont trop de contraintes sont saturées. C'est malheureusement le cas de nombreux problèmes issus de l'industrie, particulièrement du problème de partitionnement dont le taux de dégénerescence est intrinsèquement élevé.

Une autre approche de ce type de problèmes est celle des algorithmes *primaux* : il s'agit de partir d'une solution entière non optimale, de trouver une direction qui mène vers une meilleure solution entière, puis d'itérer ce processus jusqu'à atteindre l'optimalité. À chaque étape, un sous-problème d'*augmentation* est résolu : trouver une direction d'amélioration (ou d'augmentation) ou affirmer que la solution courante est optimale. Les travaux concernant les méthodes primales sont moins nombreux que ceux sur le *branch-and-bound*, qui représentent depuis quarante ans la filière dominante pour la résolution de problèmes en nombres entiers. Développer une méthode primale efficace en pratique constituerait ainsi un changement majeur dans le domaine. Des travaux computationels sur des algorithmes primaux ressortent deux principaux défis rencontrés lors de la conception et l'implémentation de ces méthodes. D'une part, de nombreuses directions d'amélioration sont *irréalisables*, c'est-à-dire qu'effectuer un pas, aussi petit soit-il, dans ces directions implique une violation des contraintes du problème. On parle alors de *dégénérescence* ; c'est par exemple le cas des directions associées à certains pivots de simplexe (pivots dégénérés). Les directions irréalisables ne permettent pas à l'algorithme de progresser et peuvent mettre en péril sa terminaison s'il est impossible de déterminer de direction réalisable. D'autre part, lorsqu'une direction d'amélioration réalisable pour la relaxation linéaire a été déterminée, il est difficile de s'assurer que la solution vers laquelle elle mène est entière. Parmi les algorithmes primaux existants, celui qui apparait comme le plus prometteur est le simplexe en nombres entiers avec décomposition (*Integral Simplex Using Decomposition*, ISUD) car il intègre au cadre primal des techniques de décomposition permettant de se prémunir des effets néfastes de la dégénerescence. Il s'agit à notre connaissance du premier algorithme de type primal capable de battre le *branch-and-bound* sur des instances de grande taille ; par ailleurs, la différence est d'autant plus importante que le problème est grand. Bien que fournissant des éléments de réponse à la problématique de la dégénerescence, cette méthode n'aborde pas pour autant la question de l'intégralité lors du passage à une solution de meilleur coût ; et pour qu'ISUD puisse envisager de supplanter les méthodes de type *branch-and-bound*, il lui faut parcourir cette deuxième moitié du chemin. Il s'agit là de l'objectif de ce doctorat : **augmenter le taux de directions entières trouvées par ISUD pour le rendre applicable aux instances industrielles de grande taille, de type planification de personnel**.

Pour aller dans cette direction, nous approfondissons tout d'abord les connaissances théoriques sur ISUD. Formuler ce dernier comme un algorithme primal, comprendre en quoi il se rattache à cette famille, le traduire pour la première fois dans un langage exclusivement primal sans faire appel à la dualité, constituent le terreau de cette thèse. Cette analyse permet ensuite de mieux décrire la géométrie sous-jacente ainsi que les domaines de réalisabilité des différents problèmes linéaires considérés. Quand bien même ce pan majeur de notre travail n'est pas présenté dans cette thèse comme un chapitre à part entière, il se situe indubitablement à l'origine de chacune de nos idées, de nos approches et de nos contributions. Cette approche de l'algorithme sous un angle nouveau donne lieu à de nombreuses simplifications, améliorations et extensions de résultats déjà connus.

Dans un premier temps, nous généralisons la formulation du problème d'augmentation afin d'augmenter la probabilité que la direction déterminée par l'algorithme mène vers une nouvelle solution entière. Lors de l'exécution d'ISUD, pour déterminer la direction qui mènera à la solution suivante, on résout un programme linéaire dont la solution est une direction d'amélioration qui appartient au cône des directions réalisables. Pour s'assurer que ce programme est borné (les directions pourraient partir à l'infini), on lui ajoute une contrainte de normalisation et on se restreint ainsi à une section de ce cône. Dans la version originelle de l'algorithme, les coefficients de cette contrainte sont uniformes. Nous généralisons cette contrainte à une section quelconque du cône et montrons que la direction réalisable déterminée par l'algorithme dépend fortement du choix des coefficients de cette contrainte ; il en va de même pour la probabilité que la solution vers laquelle elle mène soit entière. Nous étendons les propriétés théoriques liés à la décomposition dans l'algorihtme ISUD et montrons de nouveaux résultats dans le cas d'un choix de coefficients quelconques. Nous déterminons de nouvelles propriétés spécifiques à certains choix de normalisation et faisons des recommandations pour choisir les coefficients afin de pénaliser les directions fractionnaires au profit des directions entières. Des résultats numériques sur des instances de planification de personnel montrent le potentiel de notre approche. Alors que la version originale d'ISUD permet de résoudre 78% des instances de transport aérien du *benchmark* considéré, 100% sont résolues grâce à l'un, au moins, des modèles que nous proposons.

Dans un second temps, nous montrons qu'il est possible d'adapter des méthodes de plans coupants utilisés en programmation linéaire en nombres entiers au cas d'ISUD. Nous montrons que des coupes peuvent êtres transférées dans le problème d'augmentation, et nous caractérisons l'ensemble des coupes transférables comme l'ensemble, non vide, des coupes primales saturées pour la solution courante du problème de partitionnement. Nous montrons que de telles coupes existent toujours, proposons des algorithmes de séparation efficaces pour les coupes primales de cycle impair et de clique, et montrons que l'espace de recherche de ces

coupes peut être restreint à un petit nombre de variables, ce qui rend le processus efficace. Des résultats numériques prouvent la validité de notre approche ; ces tests sont effectués sur des instances de planification de personnel navigant et de chauffeurs d'autobus allant jusqu'à 1 600 contraintes et 570 000 variables. Sur les instances de transport aérien testées l'ajout de coupes primales permet de passer d'un taux de résolution de 70% à 92%. Sur de grandes instances d'horaires de chauffeurs d'autobus, les coupes prouvent l'optimalité locale de la solution dans plus de 80% des cas.

Dans un dernier temps, nous modifions dynamiquement les coefficients de la contrainte de normalisation lorsque la direction trouvée par l'algorithme mène vers une solution fractionnaire. Nous proposons plusieurs stratégies de mise-à-jour visant à pénaliser les directions fractionnaires basées sur des observations théoriques et pratiques. Certaines visent à pénaliser la direction choisie par l'algorithme, d'autres procèdent par perturbation des coefficients de normalisation en utilisant les équations des coupes mentionnées précédemment. Cette version de l'algorithme est testée sur un nouvel ensemble d'instances provenant de l'industrie du transport aérien. À notre connaissance, l'ensemble d'instances que nous proposons n'est comparable à aucun autre. Il s'agit en effet de grands problèmes d'horaires de personnel navigant allant jusqu'à 1 700 vols et 115 000 rotations, donc autant de contraintes et de variables. Ils sont posés sous la forme de problèmes de partitionnement pour lesquels nous fournissons des solutions initiales comparables à celles dont on disposerait en milieu industriel. Notre travail montre le potentiel qu'ont les algorithmes primaux pour résoudre des problèmes de planification de personnel navigant, problèmes clés pour les compagnies aériennes, tant par leur complexité intrinsèque que par les conséquences économiques et financières qu'ils entraînent.

ABSTRACT

Optimization is a three-step process. Step one *models* the problem and writes it as a *mathematical program*, i.e., a set of equations that includes an objective one seeks to minimize or maximize (typically the costs or benefit of a company) and constraints that must be satisfied by any acceptable solution (operational constraints, collective agreement, etc.). The unknowns of the model are the decision variables; they correspond to the quantities the decision-maker wants to infer. A model that perfectly represents reality is exact, otherwise it is approximate. The second step of the optimization process is the *solution* of the mathematical program, i.e., the determination of a solution that satisfies all constraints and for which the objective value is as good as possible. To this end, one generally uses an *algorithm*, a self-contained step-by-step set of operating rules that solves the problem in a finite number of operations. The algorithm is translated by means of a programming language into an executable program run by a computer; the execution of such software solves the mathematical program. Finally, the last step is the *adaptation* of the mathematical solution to reality. When the model is only approximate, the output solution may not fit the original requirements and therefore require a posteriori modifications. This thesis concentrates on the second of these three steps, the *solution* process. More specifically, we design and implement an algorithm that solves a specific mathematical program: *set partitioning*.

The set partitioning problem models a very wide range of applications: workforce scheduling, logistics, electricity production planning, pattern recognition, etc. In each of these examples, the objective function and the constraints have different physical significations but the structure of the model is the same. From a mathematical point of view, it is an integer linear program whose decision variables can only take value 0 or 1. It is linear because both the objective and the constraints are linear functions of the variables. Most algorithms used to solve this family of programs are based on the principle called *branch-and-bound*. At first, the integrality constraints are relaxed; solutions may thus be fractional. The solution of the resulting program – called *linear relaxation* of the integer program – is significantly easier than that of the integer program. Then, to recover integrality, the problem is separated to eliminate fractional solutions. From the splitting a branching tree arises, in which, at each node, the relaxation of a set partitioning problem as big as the original one is solved. The size of that tree, and thus the solving time, grows exponentially with the size of the instance. Furthermore, the algorithm that solves the linear relaxations, the *simplex*, performs poorly on degenerate problems, i.e., problems for which too many constraints are tight. It is unfortunately the case of many industrial problems, and particularly of the set partitioning

problem whose degeneracy rate is intrinsiquely high.

An alternative approach is that of primal algorithms: start from a nonoptimal integer solution and find a direction that leads to a better one (also integer). That process is iterated until optimality is reached. At each step of the process one solves an *augmentation* subproblem which either outputs an augmenting direction or asserts that the current solution is optimal. The literature is significantly less abundant on primal algorithms than on branch-and-bound and the latter has been the dominant method in integer programming for over forty years. The development of an efficient primal method would therefore stand as a major breakthrough in this field. From the computational works on primal algorithms, two main issues stand out concerning their design and implementation. On the one hand, many augmenting directions are *infeasible*, i.e., taking the smallest step in such a direction results in a violation of the constraints. This problem is strongly related to *degeneracy* and often affects simplex pivots (e.g., degenerate pivots). Infeasible directions prevent the algorithm from moving ahead and may jeopardize its performance, and even its termination when it is impossible to find a feasible direction. On the other hand, when a cost-improving direction has been succesfully determined, it may be hard to ensure that it leads to an integer solution. Among existing primal algorithms, the one appearing to be the most promising is the integral simplex using decomposition (ISUD) because it embeds decomposition techniques that palliate the unwanted effects of degeneracy into a primal framework. To our knowledge, it is the first primal algorithm to beat branch-and-bound on large scale industrial instances. Furthermore, its performances improve when the problem gets bigger. Despite its strong assets to counter degeneracy, however, this method does not handle the matter of integrality when reaching out for the next solution; and if ISUD is to compete with branch-and-bound, it is crucial that this issue be tackled. Therefore, the purpose of this thesis is the following: **increasing the rate of integral directions found by ISUD to make it fully competitive with existing solvers on large-scale industrial workforce scheduling instances**.

To proceed in that direction, we first deepen the theoretical knowledge on ISUD. Formulating it as a primal algorithm, understanding how it belongs to that family, and translating it in a purely primal language that requires no notion of duality provide a fertile ground to our work. This analysis yields geometrical interpretations of the underlying structures and domains of the several mathematical programs involved in the solution process. Although no chapter specifically focuses on that facet of our work, most of our ideas, approaches and contributions stem from it. This groundbreaking approach of ISUD leads to simplifications, strengthening, and extensions of several theoretical results.

In the first part of this work, we generalize the formulation of the augmentation problem in order to increase the likelihood that the direction found by the algorithm leads to a new integer solution. In ISUD, to find the edge leading to the next point, one solves a linear program to select an augmenting direction from a cone of feasible directions. To ensure that this linear program is bounded (the directions could go to infinity), a normalization constraint is added and the optimization is performed on a section of the cone. In the original version of the algorithm, all weights take the same value. We extend this constraint to the case of a generic normalization constraint and show that the output direction depends strongly on the chosen normalization weights, and so does the likelihood that the next solution is integer. We extend the theoretical properties of ISUD, particularly those that are related to decomposition and we prove new results in the case of a generic normalization constraint. We explore the theoretical properties of some specific constraints, and discuss the design of the normalization constraint so as to penalize fractional directions. We also report computational results on workforce scheduling instances that show the potential behind our approach. While only 78% of aircrew scheduling instances from that benchmark are solved with the original version of ISUD, 100% of them are solved by at least one of the models we propose.

In the second part, we show that cutting plane methods used in integer linear programming can be adapted to ISUD. We show that cutting planes can be transferred to the augmentation problem, and we characterize the set of transferable cuts as a nonempty subset of primal cuts that are tight to the current solution. We prove that these cutting planes always exist, we propose efficient separation procedures for primal clique and odd-cycle cuts, and we prove that their search space can be restricted to a small subset of the variables making the computation efficient. Numerical results demonstrate the effectiveness of adding cutting planes to the algorithm. Tests are performed on small- and large-scale set partitioning problems from aircrew and bus-driver scheduling instances up to 1,600 constraints and 570,000 variables. On the aircrew scheduling instances, the addition of primal cuts raises the rate of instances solved from 70% to 92%. On large bus drivers scheduling instances, primal cuts prove that the solution found by ISUD is optimal over a large subset of the domain for more than 80% of the instances.

In the last part, we dynamically update the coefficients of the normalization constraint whenever the direction found by the algorithm leads to a fractional solution, to penalize that direction. We propose several update strategies based on theoretical and experimental results. Some penalize the very direction returned by the algorithm, others operate by perturbating the normalization coefficients with those of the aforementionned primal cuts. To prove the efficiency of our strategies, we show that our version of the algorithm yields

better results than the former version and than classical branch-and-bound techniques on a benchmark of industrial aircrew scheduling instances. The benchmark that we propose is, to the best of our knowledge, comparable to no other from the literature. It provides large-scale instances with up to 1,700 flights and 115,000 pairings, hence as many constraints and variables, and the instances are given in a set-partitioning form together with initial solutions that accurately mimic those of industrial applications. Our work shows the strong potential of primal algorithms for the crew scheduling problem, which is a key challenge for large airlines, both financially significant and notably hard to solve.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

IPS	*Improved Primal Simplex*
ISUD	*Integral Simplex Using Decomposition*
SPP	*Set Partitioning Problem*
AUG	*Augmentation Problem*
iAUG	*Integral Augmentation Problem*
fAUG	*Fractional Augmentation Problem*
SEP	*Standard Separation Problem*
P-SEP	*Primal Separation Problem*

CHAPITRE 1 INTRODUCTION

Le problème de partitionnement. Le sujet principal de cette thèse est la résolution d'un problème particulier d'optimisation linéaire en nombres entiers appelé *problème de partitionnement d'ensemble*. Exprimé à l'origine comme la question de la partition d'un ensemble à coût minimal, il est aujourd'hui utilisé pour modéliser des situations rencontrées en pratique dans de nombreux domaines allant de la planification de personnel (Desrosiers et al., 1995) à la logistique (Baldacci & Mingozzi, 2009) en passant par la production d'éléctricité (Rozenknop et al., 2013), le partage équitable (Vetschera, 2010) ou encore la reconnaissance de formes (Sumetphong & Tangwongsan, 2012). L'application sur laquelle nous nous focalisons dans ce manuscrit est la planification de personnel, en particulier dans les domaines du transport aérien et des transports en commun.

Optimisation de la planification du personnel dans le transport aérien. La question des horaires d'équipages en transport aérien est traditionnellement, et encore aujourd'hui, scindée en plusieurs problèmes résolus séquentiellement. D'abord, un ensemble de rotations – suite de vols commençant et finissant au même aéroport – couvrant toutes les tâches à effectuer pour un coût minimum est généré sur un horizon cyclique d'une semaine typique. Ensuite, cette semaine est répliquée à l'identique pour obtenir un ensemble de rotations sur un mois. Si des variations doivent être ajoutées d'une semaine à l'autre, des corrections pouvant déteriorer la qualité de la solution sont apportées. Enfin, on construit les blocs mensuels des employés (suite de rotations et congés). Ces rotations et ces blocs sont soumis à des contraintes opérationnelles, à celles de la convention collective et aux standards de sécurité.

Nécessité de résoudre des problèmes plus grands. Premièrement, la taille des instances à résoudre pour la planification du personnel dans les grandes compagnies aériennes grossit de jour en jour. Le trafic aérien est en constante augmentation d'une part, et les fusions entre compagnies (Air France-KLM, Continental Airlines et United Airlines, ...) entraînent une augmentation considérable du nombre de vols et d'employés à considérer lors de la création des emplois du temps.

Ensuite, il est primordial d'adapter l'offre de vols à la demande. Déterminer une bonne offre de vols sur un horizon d'un mois est aujourd'hui tout à fait concevable. Cependant, fabriquer un emploi du temps sur un horizon mensuel n'est pas encore envisageable. Comme nous l'avons dit, l'horizon de résolution actuel est d'une semaine. L'offre de vol est ainsi

limitée à la répétition quasi-identique d'une semaine type sur le mois. À l'origine, l'horizon répliqué était de l'ordre de la journée. Le passage à un horizon d'une semaine a permis d'amener le taux de remplissage des avions d'environ 60-65% à 80-85%. On estime que considérer directement un horizon mensuel pour générer les rotations permettrait de mieux adapter la solution proposée à la demande et ainsi obtenir des taux de remplissage dépassant 90%. Dans le cas d'Air Canada, en 2013, le taux de remplissage moyen des avions s'élevait à 82,8% [1]. S'il passait à 90%, cela représenterait une augmentation de 8,0% du revenu lié à la vente de billets, soit environ 881 millions de dollars, une somme significative dans la mesure où le bénéfice de l'entreprise est typiquement de l'ordre de $\pm$100 millions de dollars par an.

Enfin, Saddoune et al. (2011) ont montré que le passage à une construction simultanée des blocs mensuels et des rotations permettrait de réduire significativement les dépenses sur certaines instances de taille moyenne (jusqu'à 5% de la masse salariale du personnel navigant). Il serait donc intéressant d'être capable d'intégrer ces étapes sur de grandes instances. À ce jour, les algorithmes existants ne permettent pas de s'attaquer à des problèmes de cette taille.

Limites avérées des algorithmes existants. Aujourd'hui, les problèmes de grande taille provenant de l'industrie ne sont pas solubles en temps raisonnable. L'algorithme le plus répandu, le *branch-and-bound*, opère de la façon suivante : dans un premier temps, les contraintes d'intégralité sont relâchées et on obtient la solution fractionnaire optimale avec l'algorithme du simplexe. Dans un second temps, on explore un arbre de branchement pour rétablir l'intégralité de la solution. La taille de cet arbre, et donc le temps d'exécution, croissent exponentiellement avec la taille des problèmes. De plus, l'algorithme utilisé pour résoudre la relaxation – le *simplexe* – éprouve des problèmes sur les problèmes dégénérés, c'est-à-dire des problèmes un nombre trop important sont saturées, ou bien, de façon équivalente, dont certaines des variables de la *base* du simplexe sont nulles. On constate que c'est le cas de nombreux problèmes issus de l'industrie, et tout particulièrement du problème de partitionnement dont le taux de dégénérescence est intrinsèquement élevé.

Un autre paradigme. Une autre approche de ce type de problèmes est celle des algorithmes *primaux* : il s'agit de partir d'une solution entière non optimale, de trouver une direction qui mène vers une meilleure solution (entière, elle aussi) puis d'itérer ce processus jusqu'à atteindre l'optimalité. A chaque étape, un sous-problème d'*augmentation* est résolu : trouver une direction d'amélioration ou affirmer que la solution courante est optimale. Les travaux concernant les méthodes primales sont moins nombreux que ceux sur le *branch-and-*

1. Voir Air Canada (2014)

bound qui représente depuis 40 ans la filière dominante pour la résolution de problèmes en nombres entiers. Développer une méthode primale efficace en pratique représenterait ainsi un changement majeur dans le domaine. Des rares travaux computationels implémentant des algorithmes primaux ressortent deux principaux défis rencontrés lors de la conception et l'implémentation de ces méthodes. D'une part, de nombreuses directions d'amélioration sont *irréalisables*, c'est-à-dire qu'effectuer un pas positif, aussi petit soit-il, dans ces directions implique une violation des contraintes du problème. On parle alors de *dégénérescence* ; c'est par exemple le cas des directions associées à certains pivots de simplexe (pivots dégénérés). Elles ne permettent pas à l'algorithme de progresser et peuvent mettre en péril la terminaison du processus dans le cas où l'algorithme n'est pas à même de proposer de direction réalisable. D'autre part, lorsqu'une direction d'amélioration *réalisable* pour la relaxation linéaire a été déterminée, il est difficile de s'assurer que la solution vers laquelle elle mène est entière.

Décomposer pour contrer la dégénérescence. Récemment, Zaghrouti et al. (2014) ont introduit le simplexe en nombres entiers avec décomposition (*Integral Simplex Using Decomposition*, ISUD) : ce nouvel algorithme primal, dont l'application est pour l'instant limitée au problème de partitionnement pur, intègre des techniques de décomposition pour se prémunir des effets néfastes de la dégénerescence. Ces techniques proviennent du domaine de la programmation linéaire (voir Elhallaoui et al. (2011)) et leur intégration dans ISUD a permis de réduire significativement le temps de résolution de problèmes de grande taille. Fournissant des éléments de réponse à la problématique de la dégénerescence, cette méthode n'aborde pas pour autant la question de l'intégralité lors du passage à une solution de meilleur coût ; et pour qu'ISUD puisse envisager de supplanter les méthodes de type *branch-and-bound*, il lui faut parcourir cette deuxième moitié du chemin. Il s'agit là de l'objectif de ce doctorat : encourager et assurer l'intégralité à chaque étape de l'algorithme ISUD, d'abord dans le cas du problème de partitionnement, et ensuite dans un cas plus général.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.1 L'*Optimisation mathématique* : les mathématiques au service du réel

L'*optimisation mathématique* est la discipline scientifique ayant pour but la résolution de problèmes concrets, généralement issus des milieux industriels ou économiques, grâce à l'application de méthodes mathématiques et algorithmiques. Le processus d'optimisation se décompose en trois étapes : modélisation, résolution et adaptation.

1. *Modéliser* : La première étape consiste à représenter le problème sous la forme d'un *programme mathématique*, ensemble d'équations constitué d'un objectif à minimiser ou maximiser (typiquement, les coûts ou le bénéfice de l'entreprise) et de contraintes à satisfaire (contraintes opérationnelles, convention collective, etc.). Aux décisions à prendre correspondent les variables du problème. Ce modèle peut se révéler exact, ou alors approximer la situation rencontrée en pratique. En recherche opérationnelle, un programme mathématique s'écrit sous la forme suivante :

$$z^\star_{\text{OPT}} = \min \{ f(x) \,|\, \boldsymbol{x} \in X \} \tag{OPT}$$

   où $\boldsymbol{x}$ est le vecteur des variables, $\mathcal{F}_{\text{OPT}} = X$ le domaine, et $f : X \to \mathbb{R}$ la fonction objectif de OPT. Il s'agit ici d'un problème de minimisation (min), mais on pourrait tout aussi bien considérer un problème de maximisation (max). Tout élément $\boldsymbol{x} \in X$ est appelé solution réalisable du problème, et si $\boldsymbol{x}$ réalise le minimum $z^\star_{\text{OPT}}$ de f sur le domaine X, on dit que $\boldsymbol{x}$ est une solution optimale. Le modèle est « parfait » s'il est à la fois exact et s'il existe une méthode efficace pour le résoudre. Cependant, de tels modèles ne sont que très rarement disponibles pour les problèmes industriels. Dans certains cas, on peut même prouver qu'il n'en existe pas !
2. *Résoudre* : Ensuite, une solution de ce programme mathématique est déterminée, généralement grâce à l'outil informatique. Pour ce faire, on applique généralement un *algorithme de résolution*, ensemble de règles opératoires dont l'application permet de résoudre le problème énoncé au moyen d'un nombre fini d'opérations. Un algorithme peut être traduit grâce à un langage de programmation en un programme exécutable par un ordinateur. Idéalement, la solution ainsi déterminée est optimale, mais du fait de la complexité et de la taille de certaines instances, il n'est pas toujours possible de résoudre le modèle à l'optimum avec les moyens dont nous disposons aujourd'hui. La plupart des algorithmes de résolution dépendent du type de problème considéré,

de la nature de la fonction objectif (continue, différentiable, convexe, linéaire, etc.) et du type de contraintes (égalités/inégalités, variables non négatives/entières/binaires, etc.). De même, la performance des algorithmes de résolution dépend de la nature du problème et de la taille des instances considérées.

3. *Ajuster* : La solution déterminée est finalement confrontée aux contraintes réelles et soumises aux instances décidantes (généralement différentes de celles qui modélisent et résolvent les problèmes). Si le modèle n'est pas exact et que la solution n'est pas concrètement implantable, soit une rétroaction sur le modèle est effectuée et celui-ci est modifié pour prendre plus de contraintes en compte, soit la solution est ajustée pour se conformer aux exigences pratiques.

Le livre de Bonnans & Gaubert (2010) constitue une excellent introduction aux domaines de la recherche opérationnelle et de l'optimisation mathématique, ainsi qu'aux différents types de programmes d'optimisation et algorithmes les plus répandus.

2.2 Notations

Dans ce mémoire, nous utilisons les notations et conventions suivantes. $\mathbb{R}$, $\mathbb{Z}$ et $\mathbb{N}$ décrivent respectivement l'ensemble des réels, des entiers relatifs et naturels. Les matrices et les vecteurs sont respectivement notés en caractères majuscules et minuscules, en gras ; les ensembles d'indices sont notés en caractères majuscules cursifs. Étant donné une matrice $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ et deux indices $i \in \{1,\dots,m\}$ et $j \in \{1,\dots,n\}$, $\boldsymbol{A}_{i\boldsymbol{\cdot}}$ et $\boldsymbol{A}_{\boldsymbol{\cdot}j}$ désignent respectivement la ligne et la colonne de $\boldsymbol{A}$ d'indice i et j. Pour tous ensembles d'indices $\mathcal{I} \subseteq \{1,\dots,m\}$ et $\mathcal{J} \subseteq \{1,\dots,n\}$, $\boldsymbol{A}_{\mathcal{I}\boldsymbol{\cdot}}$ décrit la matrice composée des lignes de $\boldsymbol{A}$ indexées par $\mathcal{I}$, et $\boldsymbol{A}_{\boldsymbol{\cdot}\mathcal{J}}$ celle composée des colonnes de $\boldsymbol{A}$ indexées par $\mathcal{J}$. De même, $\boldsymbol{A}_{\mathcal{I}\mathcal{J}}$ décrit la sous-matrice de $\boldsymbol{A}$ dont les indices des lignes sont indexés par $\mathcal{I}$ et ceux des colonnes par $\mathcal{J}$. Pour tout vecteur $\boldsymbol{v} \in \mathbb{R}^n$, v_j est la $j^{\text{ème}}$ coordonnée de $\boldsymbol{v}$ et $\boldsymbol{v}_{\mathcal{J}}$ est le vecteur formé des v_j, $j \in \mathcal{J}$. $\mathbf{0}$ et $\boldsymbol{e}$ représentent respectivement le vecteur de zéros et de uns dont la dimension s'adapte au contexte. $\boldsymbol{A}^T$ et $\boldsymbol{v}^T$ désignent respectivement la transposée de $\boldsymbol{A}$ et celle de $\boldsymbol{v}$. Enfin, pour tout ensemble de vecteurs (ou matrice) $\boldsymbol{V} = (\boldsymbol{v}_1,\dots,\boldsymbol{v}_k)$ de $\mathbb{R}^n$, l'espace vectoriel des combinaisons linéaires de vecteurs de $\boldsymbol{V}$, appelé espace engendré par $\boldsymbol{V}$, est noté $\mathsf{Span}\,(\boldsymbol{V}) = \left\{\boldsymbol{\lambda}^T\boldsymbol{V} \mid \boldsymbol{\lambda} \in \mathbb{R}^k\right\}$.

Étant donné un ensemble $E \subseteq \mathbb{R}^n$, le plus petit ensemble convexe de $\mathbb{R}^n$ qui le contient, appelé enveloppe convexe de E, est noté $Conv(E)$. Lorsque E est un polyèdre, l'ensemble de ses points extrêmes est noté $Ext\,(E)$. Pour tout programme d'optimisation P, la valeur optimale de P est notée $z_P^\star$, et son domaine réalisable $\mathcal{F}_{\mathrm{P}}$. Sauf indication du contraire, les

programmes d'optimisation sont tous présentés sous leur forme de minimisation et leurs domaines supposés bornés.

2.3 Programmation linéaire

2.3.1 Définitions

Parmi les formes de programmes mathématiques les plus simples à résoudre se trouvent les *programmes linéaires* : l'objectif f est une fonction linéaire, et les contraintes définissant le domaine sont des (in)égalités linéaires. Pour une introduction simple et détaillée à la programmation linéaire, le lecteur pourra se référer à Vanderbei (2001). Si le nombre de variables de décision est n et le nombre de contraintes linéaires m, un tel programme sous sa forme dite *standard* s'écrit

$$z^\star_{LP} = \min \left\{ \boldsymbol{c}^T \boldsymbol{x} \,|\, \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0} \right\}, \tag{LP}$$

où $\boldsymbol{c} \in \mathbb{R}^n$ s'appelle le vecteur des coûts, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ la matrice des contraintes, et $\boldsymbol{b} \in \mathbb{R}^m$. Tout programme d'optimisation comportant une ou plusieurs inégalité(s) linéaire(s) peut être transformé en un programme sous forme standard ; il n'est donc pas restrictif de ne considérer que des problèmes sous cette forme. La matrice $\boldsymbol{A}$ est supposée de plein rang, et $m \leq n$. Les équations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ sont appelées contraintes linéaires, et $\boldsymbol{x} \geq \boldsymbol{0}$ les contraintes de non négativité. Le domaine réalisable de LP, $\mathcal{F}_{\text{LP}} = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0\}$, est un polyèdre et, quand il est borné, c'est un polytope. De la nature linéaire des contraintes et de l'objectif, il est possible de déduire de nombreuses propriétés sur la nature de l'ensemble des solutions. En particulier, si $z^\star_{LP} < +\infty$, on dit que LP est borné, et il existe une solution optimale qui est un point extrême du polyèdre $\mathcal{F}_{\text{LP}}$.

2.3.2 Algorithme du simplexe et dégénérescence

Les deux familles d'algorithmes les plus couramment utilisées en programmation linéaire sont les méthodes de points intérieurs et les méthodes basées sur le simplexe. Bien que les points intérieurs possèdent d'indéniables avantages sur certains types de problèmes, le simplexe reste un algorithme couramment utilisé pour résoudre une grande partie des problèmes rencontrés en pratique notamment dans les schémas de décomposition et les algorithmes d'énumération pour les problèmes en nombres entiers. Dans ce manuscrit, nous nous intéresserons en priorité aux algorithmes de type simplexe, et plus précisément au *simplexe primal* originellement introduit par Dantzig en 1947. Pour une description précise de l'al-

gorithme, de ses avantages, de ses limites, et de ses implémentations possibles, le lecteur pourra se référer à l'ouvrage de Maros (2003). Le simplexe repose sur la notion de *base* : une base est un ensemble $\mathcal{B}$ d'indices de colonnes de $\boldsymbol{A}$ de cardinal m, tel que l'ensemble des colonnes correspondantes $\{\boldsymbol{A}_{\cdot j}\}_{j\in\mathcal{B}}$ est linéairement indépendant. L'ensemble des indices des colonnes qui ne font pas partie de la base est noté $\mathcal{H}$. À $\mathcal{B}$ est associée une unique solution de base, $\boldsymbol{x} = (\boldsymbol{x}_\mathcal{B}, \boldsymbol{x}_\mathcal{H}) = \left(\boldsymbol{A}_{\cdot\mathcal{B}}^{-1}\boldsymbol{b}, \boldsymbol{0}\right)$; deux bases distinctes pouvant engendrer la même solution. L'ensemble des solutions de base est exactement l'ensemble des points extrêmes du polytope définit par les contraintes, $Ext\left(\mathcal{F}_{\mathrm{LP}}\right)$. Étant donné une solution initiale $\boldsymbol{x}^0 \in Ext\left(\mathcal{F}_{\mathrm{LP}}\right)$, le simplexe fournit une suite de solutions $(\boldsymbol{x}^k)_{k=0,\dots,K}$ de coût non croissant dont la dernière, $\boldsymbol{x}^K$, est optimale. De plus, ces solutions sont toutes des sommets de $\mathcal{F}_{\mathrm{LP}}$ et pour tout $k \in \{0,\dots,(K-1)\}$, $\boldsymbol{x}^k$ et $\boldsymbol{x}^{k+1}$ sont soit confondus, soit des sommets voisins du polytope. Du point de vue géométrique, le chemin parcouru par l'algorithme suit ainsi les arêtes du polyèdre.

Algébriquement, le passage d'une solution de base à la suivante consiste en l'échange d'un des indices de $\mathcal{B}$ avec un de ceux de $\mathcal{H}$. Cette opération s'appelle un pivot de simplexe. La variable de $\mathcal{H}$ sélectionnée pour entrer dans la base, appelée *variable entrante*, est de coût réduit négatif (minimisation). C'est-à-dire que si elle prend une valeur positive à la nouvelle solution, la valeur de l'objectif est améliorée. La taille de la base étant constante, ajouter une nouvelle variable en base implique nécessairement d'en retirer une autre, appelée *variable sortante*, dont la valeur à la nouvelle solution est nulle. Étant donnée une variable entrante, il peut parfois exister plusieurs variables sortantes potentielles. À la nouvelle base $\mathcal{B}'$ correspond la nouvelle solution $\boldsymbol{x}' = (\boldsymbol{x}_{\mathcal{B}'}, \boldsymbol{x}_{\mathcal{H}'}) = \left(\boldsymbol{A}_{\cdot\mathcal{B}'}^{-1}\boldsymbol{b}, \boldsymbol{0}\right)$. Si $\boldsymbol{x}' \neq \boldsymbol{x}$, on dit que le pivot est non dégénéré, et alors $\boldsymbol{x}$ et $\boldsymbol{x}'$ sont des sommets voisins de $\mathcal{F}_{\mathrm{LP}}$. Si certaines variables (de base) de $\boldsymbol{x}_\mathcal{B}$ sont nulles, $\boldsymbol{x}$ est une solution dite *dégénérée*, et on peut avoir $\boldsymbol{x}' = \boldsymbol{x}$; on parle alors de pivot dégénéré. Le nombre de bases représentant le même sommet du polytope peut être très élevé. Si $m = 100$ et $n = 1000$ et que 50 des variables de $\boldsymbol{x}_\mathcal{B}$ sont nulles, on peut remplacer chacune d'elle par n'importe laquelle des variables hors base sans changer la solution associée. Le nombre de bases représentant cette solution atteint alors $C_{950}^{50} \approx 6.8\ 10^{83}$. Le phénomène de dégénérescence apparaît souvent, en particulier dans les instances issues de problèmes industriels, et peut compromettre sinon la terminaison du moins la performance de l'algorithme.

Nombre de techniques ont été proposées pour pallier les effets de la dégénérescence. Parmi celles-ci, certaines se penchent plus précisément sur la méthode de détermination de la variable entrante. Ainsi, Greenberg (1978) associe à chaque variable hors base une quantité réelle qui, si elle est positive, indique qu'entrer cette variable en base mènera à un pivot dégénéré. Le nombre d'opérations nécessaires au calcul de ces valeurs est similaire à celui

du calcul d'un coût réduit, mais ce test reste heuristique car il ne permet pas d'identifier toutes les variables dont les pivots associés sont dégénérés. D'autres techniques telles que les perturbations (Benichou et al., 1977) ou le décalage de bornes (*bound shifting*, Gill et al. (1989)) ont été proposées. Elles se basent sur des modifications aléatoires de la valeur des variables dégénérées ou de leurs bornes pour mettre fin à une série de pivots sans amélioration. Ces modifications aléatoires permettent d'éviter de faire des pivots non améliorants mais ne font généralement que les remplacer par une suite quasiment équivalente de petites améliorations. Bien que ces méthodes ne reposent pas toujours sur des bases théoriques solides, elles permettent cependant d'obtenir de bons résultats en pratique et sont intégrées à la plupart des implémentations de l'algorithme du simplexe. Pour davantage d'informations sur la dégénérescence, et en particulier sur ses aspects algorithmiques et informatiques, le lecteur pourra se référer à l'excellente analyse de Maros (2003).

2.3.3 L'agrégation de contraintes

Une autre approche face à un problème dégénéré consiste à tirer profit de cette caractéristique plutôt que la subir. En dimension n, un sommet dégénéré du polytope des contraintes est le point d'intersection de plus de n contraintes linéaires (en incluant les contraintes de bornes) : la dégénérescence peut donc s'interpréter comme un excès local d'information. Comme les m contraintes linéaires d'un programme sous forme standard sont toujours saturées, et que les $n - m$ variables de $\mathcal{H}$ ont une valeur nulle dans la solution de base, dès lors qu'une variable en base est nulle, la solution est dégénérée. En supposant qu'il est possible de s'affranchir localement de cet excès d'information, le problème à résoudre pour trouver une amélioration devrait donc être plus petit si la solution courante est dégénérée. Bien que prometteuse, cette approche est moins populaire que les précédentes et la littérature qui lui est consacrée peu abondante. Perold (1980) introduit une structure de dégénérescence dans la décomposition LU de $\boldsymbol{A}_{.\mathcal{B}}$ qui réduit le nombre d'opérations à effectuer lorsqu'un pivot est dégénéré. Pan (2008) étend la notion de base à celle de *base déficiente* (*deficient basis*) qui ne contient que p colonnes indépendantes, où p est strictement inférieur au nombre de lignes de la matrice $\boldsymbol{A}$. Lorsque la solution courante est dégénérée, les calculs nécessaires pour réaliser le pivot sont effectués sur une base déficiente ne contenant que p colonnes, soit moins qu'une base de simplexe. Parmi les bases déficientes possibles associées à une solution $\boldsymbol{x}$, on appelle *base de travail* (*working basis*) celle qui contient exactement l'ensemble des indices des variables non nulles $\mathcal{B}_{\mathcal{W}} = \{j \in \{1, \ldots, m\} \mid x_j > 0\}$.

En cas de dégénérescence, la redondance d'information peut s'exprimer en termes du nombre de contraintes saturées dont le point d'intersection définit la solution courante. Dans le

but de faire disparaître ce surplus d'information, Elhallaoui et al. (2011) modifient le simplexe et proposent le simplexe primal amélioré (*Improved Primal Simplex*, IPS). IPS se base sur des techniques d'agrégation dynamique des contraintes développées à l'origine pour des problèmes linéaires spécifiques (Elhallaoui et al., 2005, 2010). Il s'agit de déterminer un ensemble de contraintes redondantes dans la définition de la solution courante, et de les exclure temporairement du problème. Non seulement IPS réduit le nombre de contraintes du problème, mais il fournit aussi une décomposition qui en réduit le nombre de variables et priorise les pivots non dégénérés, exploitant ainsi doublement la dégénérescence de la solution courante. Après que tous les pivots intéressants ont été effectués dans le problème *réduit*, un problème *complémentaire* est résolu pour déterminer la direction *réalisable* de coût minimum. Une direction $\boldsymbol{d}$ est dite réalisable en $\boldsymbol{x}$ si et seulement s'il existe un pas strictement positif $\rho > 0$ tel que $(\boldsymbol{x}^0 + \rho\boldsymbol{d}) \in \mathcal{F}_{\text{LP}}$. Si une seule des variables nulles en $\boldsymbol{x}$ prend une valeur non nulle dans cette direction, celle-ci correspond à un pivot de simplexe pour lequel cette variable est entrante. Si le coût minimum d'une direction réalisable est nul, la solution courante est optimale. Dans le cas contraire, faire entrer successivement les variables non nulles d'une solution (extrême) optimale du problème complémentaire dans la base de travail $\mathcal{B}_{\mathcal{W}}$ est suffisant pour améliorer strictement la solution courante. Metrane et al. (2010) démontrent que la décomposition effectuée dans IPS est équivalente à celle de certains algorithmes de génération de colonnes en identifiant respectivement les problèmes réduit et complémentaire au problème maître et au sous-problème de la génération de colonnes. Une description tant concise que complète de IPS est donnée par Omer et al. (2015).

2.4 Programmation linéaire en nombres entiers

2.4.1 Définitions et propriétés de base

Avant de décrire les différentes méthodes de résolution de problèmes linéaires en nombres entiers, nous présentons cette branche de l'optimisation, ainsi que deux de ses cas particuliers : la programmation linéaire binaire et le problème de partitionnement.

Programmation linéaire en nombres entiers

Dans le contexte pratique, les variables de décision ne représentent pas toujours des quantités continues ou fractionnaires. Si l'on souhaite par exemple optimiser la configuration de la cabine d'un avion, on cherchera un nombre entier de places en première classe, en classe affaire et en classe économique et une réponse fractionnaire n'aurait aucun sens en pratique. Devant une telle contrainte physique, le modèle de programmation linéaire LP doit être mo-

difié pour prendre en compte l'intégralité des variables. On parle alors de *programmation linéaire en nombres entiers* et l'expression du modèle mathématique correspondant est

$$z^{\star}_{\text{PLNE}} = \min \left\{ \boldsymbol{c}^T \boldsymbol{x} \,|\, \boldsymbol{x} \in \mathbb{N}^n, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0} \right\}, \tag{PLNE}$$

où les entrées de $\boldsymbol{A}$, $\boldsymbol{c}$ et $\boldsymbol{b}$ prennent des valeurs entières. L'ensemble des solutions réalisables de ce programme mathématique $\mathcal{F}_{\text{PLNE}}$ est un ensemble discret, ne contenant que des points entiers. Le programme linéaire obtenu à partir de PLNE en supprimant les contraintes d'intégralité s'appelle sa *relaxation linéaire* (*linear relaxation*, LR) et se note PLNE^{LR}. Comme pour la programmtion linéaire, on supposera dans ce manuscrit que les ensembles de solutions $\mathcal{F}_{\text{PLNE}}$ et $\mathcal{F}_{\text{PLNE}^{\text{LR}}}$ sont bornés. Le domaine de la relaxation linéaire est donc un polytope, noté $\mathcal{F}_{\text{PLNE}^{\text{LR}}}$. Il existe de nombreux ouvrages traitant de la programmation linéaire en nombres entiers, parmi lesquels on peut citer celui de Wolsey & Nemhauser (2014), en anglais, ou encore le chapitre du livre de Bonnans & Gaubert (2010) sur ce sujet, en français.

Programmation Linéaire Binaire

Un premier cas particulier de programmation linéaire en nombres entiers est celui où les variables ne peuvent prendre que les valeurs 0 ou 1. Son rôle important est dû aux deux raisons suivantes : d'une part, tout programme linéaire en nombres entiers peut être reformulé comme un programme en variables binaires ; d'autre part, les variables binaires sont souvent nécessaires car elles permettent de modéliser la réponse à une question fermée. Le programme mathématique correspondant s'écrit

$$z^{\star}_{PLB} = \min \left\{ \boldsymbol{c}^T \boldsymbol{x} \,|\, \boldsymbol{x} \in \mathbb{N}^n, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{e} \right\}, \tag{PLB}$$

où les entrées de $\boldsymbol{A}$, $\boldsymbol{b}$ et $\boldsymbol{c}$ prennent des valeurs entières. Du point de vue théorique, le modèle PLB présente des particularités importantes. Tout d'abord, PLB ainsi que sa relaxation linéaire PLB^{LR} sont toujours bornés. Ensuite, toutes les solutions entières (binaires) sont des points extrêmes du polytope de la relaxation linéaire. En effet, chacune de ces solutions sature n contraintes de bornes ($\boldsymbol{0} \leq \boldsymbol{x}$ et $\boldsymbol{x} \leq \boldsymbol{e}$). La propriété précédemment énoncée s'écrit donc $\mathcal{F}_{\text{PLB}} \subseteq \mathit{Ext}\left(\mathcal{F}_{\text{PLB}^{\text{LR}}}\right)$. Enfin, ces problèmes ont tendance à être fortement dégénérés car un grand nombre de contraintes de bornes sont saturées. La dégénérescence est encore plus importante aux points entiers car ces derniers sont l'intersection de n contraintes de bornes saturées (une par variable), et des m contraintes d'égalité $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

Problème de Partitionnement

La dernière famille de problèmes que nous présentons ici est le *problème de partitionnement d'ensemble* (*set partitioning problem*). Comme nous l'avons dit dans l'introduction, il s'agit d'un modèle important en pratique car il permet de résoudre des problèmes dans de très nombreux domaines tels que la planification d'emplois du temps (Marsten & Shepardson, 1981), la logistique (Baldacci & Mingozzi, 2009), la production d'éléctricité (Rozenknop et al., 2013), le partage équitable (Vetschera, 2010), ou la reconnaissance de formes (Sumetphong & Tangwongsan, 2012). Introduit pour la première fois par Garfinkel & Nemhauser (1969), ce problème peut se formuler en termes ensemblistes comme suit :

SPP	Étant donné un ensemble $\mathcal{X}$ et des sous-ensembles, $\mathcal{X}_1, \ldots, \mathcal{X}_n \subseteq \mathcal{X}$ de coûts respectifs $c_1, \ldots, c_n$, former une partition de $\mathcal{X}$ composée d'éléments de $\{\mathcal{X}_i\}_{1 \leq i \leq n}$ et de coût minimal ou prouver qu'aucune n'existe.

En supposant $\mathcal{X} = \{1, \ldots, m\}$, pour $i \in \{1, \ldots, m\}$ et $j \in \{1, \ldots, n\}$, la matrice $\boldsymbol{A}$ est définie par : $A_{ij} = 1$ si $i \in \mathcal{X}_j$, 0 sinon. Ainsi, $x_j = 1$ si $\mathcal{X}_j$ est dans la partition choisie. Le membre de droite est $\boldsymbol{e}$, car chaque élément de $\mathcal{X}$ doit être représenté une et une seule fois dans la partition. Ainsi, en utilisant le vocabulaire de la programmation mathématique, le problème de partitionnement s'écrit

$$z^\star_{\text{SPP}} = \min \left\{ \boldsymbol{c}^T \boldsymbol{x} \,|\, \boldsymbol{x} \in \mathbb{N}^n, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{e}, \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{e} \right\}, \tag{SPP}$$

où $\boldsymbol{A}$ est une matrice binaire de taille $m \times n$, et $\boldsymbol{c} \in \mathbb{N}^n$ un vecteur d'entiers. En plus de toutes celles de PLB, SPP possède la propriété de *quasi-intégralité* (Trubin, 1969) : toutes les arêtes de $Conv(\mathcal{F}_{\text{SPP}})$ sont des arêtes de $\mathcal{F}_{\text{SPP}^{\text{LR}}}$. Un déplacement selon les arêtes de l'enveloppe convexe de l'ensemble réalisable qui relie un point entier à un autre point entier est donc aussi un déplacement le long des arêtes du polytope de la relaxation linéaire ; or, passer d'un sommet de $\mathcal{F}_{\text{SPP}^{\text{LR}}}$ à un de ses voisins se fait grâce à un ou plusieurs pivots de simplexe. On peut donc passer d'une solution entière à n'importe quelle autre grâce à une suite de pivots de simplexe. Pour une introduction poussée sur le problème de partitionement, et sur la théorie polyédrale associée, ainsi que sur les grandes lignes de certains algorithmes de résolution, le lecteur pourra consulter la revue de Balas & Padberg (1976) qui, bien qu'assez ancienne, reste d'actualité à de nombreux égards.

2.4.2 Classification des méthodes de résolution pour des programmes linéaires en nombres entiers

La méthode la plus répandue pour la résolution de programmes linéaires en nombres entiers est la séparation-et-évaluation (*branch-and-bound*) introduite pour la première fois par Dakin (1965). Le principe de cette méthode repose sur l'exploration récursive d'un arbre de branchement pour déterminer une solution optimale de $PLNE$. Bien qu'il soit possible de couper des branches de l'arbre en calculant des bornes inférieures/supérieures grâce à la relaxation linéaire et bien que l'ajout de coupes permette d'améliorer la relaxation et d'accélérer ainsi le processus – on parle alors de *branch-and-cut* – la taille de l'arbre de branchement est une fonction exponentielle du nombre de variables et l'algorithme peut être extrêmement lent, particulièrement sur des instances de grande taille. De plus, dans la plupart des cas rencontrés en pratique, le *branch-and-bound* n'est pas capable de tirer parti des solutions réalisables connues ni de les utiliser comme point de départ. Au vu de ces observations, il semble nécessaire de chercher des méthodes alternatives au *branch-and-bound* dans l'optique de résoudre des programmes linéaires en nombres entiers de grande taille. Bien évidemment, nous ne sommes pas les premiers – ni bien entendu les derniers – à soulever cette question et il existe d'autres familles d'algorithmes basées sur la programmation linéaire. Parmi elles, les *algorithmes primaux* feront l'objet d'une attention particulière dans ce manuscrit.

Comme indiqué par Letchford & Lodi (2002), on peut classer les algorithmes pour la programmation linéaire en nombres entiers en trois grandes familles : les méthodes dual-fractionnaires, dual-entières, et primales. Les algorithmes dual-fractionnaires maintiennent l'optimalité ainsi que la réalisabilité des contraintes linéaires à chaque itération et s'arrêtent à l'atteinte de l'intégralité. Les procédures de plans coupants telles que celle de Gomory (1958) en sont un exemple typique. De même, le schéma classique d'énumération implicite repose sur une approche dual-fractionnaire, en particulier lors de la détermination des bornes inférieures. Les méthodes dual-entières garantissent l'intégralité ainsi que l'optimalité (duale) à tout moment et s'arrêtent lorsque toutes les contraintes linéaires sont satisfaites. Letchford & Lodi n'en donnent que l'exemple d'un algorithme de Gomory (1963). Enfin, les algorithmes primaux maintiennent la réalisabilité tant linéaire qu'entière tout au long du processus et ne s'arrêtent que lorsque l'optimalité est atteinte. Ces algorithmes, au cœur de cette thèse de doctorat, sont présentés en détails à la section 2.5. Toutes ces méthodes s'appliquent à la résolution de PLNE, et en particulier à celle de PLB et SPP.

2.5 Algorithmes primaux

2.5.1 État de l'art

Comme indiqué ci-dessus, les méthodes primales sont des algorithmes de descente, pour lesquels la suite de solutions retournée $\left(\boldsymbol{x}^k\right)_{k=0,\dots,K}$ satisfait les conditions suivantes :

- C1 $\boldsymbol{x}^k \in \mathcal{F}_{\text{PLNE}}$;
- C2 $\boldsymbol{x}^K$ est optimal pour $PLNE$;
- C3 $\boldsymbol{c}^T\boldsymbol{x}^{k+1} < \boldsymbol{c}^T\boldsymbol{x}^k$.

Il faut remonter au début des années 1960 et aux travaux originels de Ben-Israel & Charnes (1962) et Young (1965) pour trouver les premières traces de méthodes primales dans la littérature, ainsi que leurs premières améliorations (Glover, 1968; Young, 1968). Dans l'algorithme d'Young (1965, 1968), à l'itération k, un pivot de simplexe donné est évalué : s'il mène à une solution entière, il est effectué ; sinon, des coupes sont générées et ajoutées au problème, changeant ainsi la structure sous-jacente des contraintes. Young définit aussi le concept de vecteur d'augmentation, ou d'amélioration, en $\boldsymbol{x}^k$, c'est-à-dire un vecteur $\boldsymbol{z} \in \mathbb{R}^n$ tel que $\boldsymbol{x}^k + \boldsymbol{z}$ appartient à $\mathcal{F}_{\text{PLNE}}$ et est de coût strictement meilleur que $\boldsymbol{x}^k$. À partir de cette notion, on définit le problème d'augmentation entière (*integral augmentation*) comme suit :

iAUG	Déterminer un vecteur d'augmentation $\boldsymbol{z} \in \mathbb{N}^n$ tel que $(\boldsymbol{x}^k + \boldsymbol{z}) \in \mathcal{F}_{\text{PLNE}}$ et $\boldsymbol{c}^T\boldsymbol{z} < 0$, ou bien prouver que $\boldsymbol{x}^k$ est optimal pour PLNE.

Pour plus de clarté, dans tout ce manuscrit, nous distinguons le problème d'augmentation entière **iAUG** de celui d'augmentation fractionnaire **fAUG** (*fractional augmentation*). **fAUG** est la relaxation de **iAUG** pour laquelle $\boldsymbol{z}$ peut être fractionnaire et $\boldsymbol{x}^k + \boldsymbol{z}$ appartient au domaine $\mathcal{F}_{\text{PLNE}^{\text{LR}}}$ de la relaxation linéaire et non nécessairement à $\mathcal{F}_{\text{PLNE}}$.

Remarque. Dans la plupart des travaux sur l'agrégation de contraintes, les problèmes sont présentés sous leur forme de minimisation, alors que les auteurs décrivent généralement les algorithmes primaux pour des problèmes de maximisation. Nous attirons donc l'attention du lecteur sur le point suivant : pour se conformer à la dénomination usuelle, nous appellons le problème d'amélioration **iAUG**, bien qu'il fournisse une direction de descente. De même, nous utilisons le terme d'augmentation pour décrire une amélioration – terme générique qui correspond dans le cas présent (minimisation) à une décroissance.

Depuis le milieu des années 1990, on assiste à un regain d'intérêt pour les algorithmes primaux, entre autres sous l'impulsion de Robert Weismantel. De nombreux travaux récents concernent spécifiquement le problème PLB, et leurs résultats sont principalement théoriques : convergence, vitesse de convergence, complexité théorique, etc. Ainsi, Schulz et al. (1995) montrent qu'en termes de complexité, le problème **iAUG** est aussi compliqué que le problème d'optimisation original : il ne faut donc pas s'attendre à trouver de méthode « miracle » pour résoudre **iAUG** en temps polynomial. Seuls de rares travaux proposent des méthodes de résolution numériques pour **iAUG** et la plupart le considèrent résolu par un oracle. Les travaux de Firla et al. (2001) et de Letchford & Lodi (2003a) sont deux des seuls exemples de résolution numérique. Pour une revue plus détaillée de la littérature existante concernant les algorithmes primaux, et en particulier de la théorie associée, le lecteur pourra consulter le texte de Spille & Weismantel (2005).

2.5.2 Coupes duales et primales

Définition 1. Soit un ensemble $E \subseteq \mathbb{R}^n$, un point $\boldsymbol{x}^\star \in \mathbb{R}^n$ et une inégalité $(\Gamma) : \boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta$, avec $\boldsymbol{\alpha} \in \mathbb{R}^n$, $\beta \in \mathbb{R}$. L'inégalité (Γ) est appelée *inégalité valide* pour E si elle est satisfaite par tous les points de E, i.e., si $\forall \boldsymbol{y} \in E, \boldsymbol{\alpha}^T\boldsymbol{y} \leq \beta$. En outre, si elle est violée par $\boldsymbol{x}^\star$ ($\boldsymbol{\alpha}^T\boldsymbol{x}^\star > \beta$), on dit qu'elle *sépare* $\boldsymbol{x}^\star$ de E. Lorsque le contexte est clair on dit alors que (Γ) est une *coupe* (ou un *plan coupant*).

La notion de coupe est essentielle en programmation en nombres entiers. En pratique, on dispose généralement d'une solution fractionnaire $\boldsymbol{x}^\star$ de la relaxation linéaire PLNE^{LR}, et on cherche à déterminer une inégalité valide pour le domaine entier $\mathcal{F}_{\text{PLNE}}$ et violée par $\boldsymbol{x}^\star$. On ajoute alors cette inégalité aux contraintes linéaires de la relaxation pour l'améliorer. Le problème consistant à déterminer si une telle inégalité existe s'appelle le *problème de séparation*, et se formule comme suit.

SEP	Étant donné un problème d'optimisation P de domaine réalisable $\mathcal{F}_{\text{P}} \subseteq \mathbb{R}^n$, et un point $\boldsymbol{x}^\star \in \mathbb{R}^n$, déterminer une inégalité valide qui sépare $\boldsymbol{x}^\star$ de $\mathcal{F}_{\text{P}}$ ou prouver qu'il n'en existe pas.

Pour résoudre un problème de type PLNE à l'aide de plans coupants, on procède généralement de la façon suivante : (1) on détermine une solution $\boldsymbol{x}^\star \in \mathcal{F}_{\text{PLNE}^{\text{LR}}}$ optimale pour la relaxation linéaire PLNE^{LR}, (2) si cette solution est entière, elle est optimale pour PLNE (non relâché), sinon on détermine un plan coupant qui sépare $\boldsymbol{x}^\star$ de $\mathcal{F}_{\text{PLNE}^{\text{LR}}}$ et (3) on ajoute l'inégalité correspondante à la relaxation. Ce processus est répété jusqu'à l'obtention d'une solution entière qui est alors optimale pour PLNE. Lorsqu'on souhaite déterminer un plan

coupant, il faut généralement choisir entre des méthodes lentes mais qui garantissent de déterminer une coupe s'il en existe et des méthodes rapides qui génèrent des coupes efficaces mais ne garantissent pas d'en trouver. Il existe des algorithmes de séparation qui assurent à la fois que l'algorithme termine et qu'il atteint une solution optimale de PLNE, telles que les coupes de Gomory, ou les *blossom inequalities* pour le problème de couplage parfait à coût minimal (Edmonds, 1965). Dans le cas du problème de partitionnement, de nombreuses familles d'inégalités valides permettent de résoudre le problème de séparation. Des thèses leur ont été consacrées (Groiez, 2013).

Lorsqu'on utilise une méthode primale, on connait une solution réalisable $\boldsymbol{x}^k \in \mathcal{F}_{\text{PLNE}}$ – la solution courante – et on cherche une direction d'amélioration réalisable à partir de cette solution. Il est relativement aisé de déterminer une direction d'amélioration pour la relaxation linéaire, mais celle-ci peut pointer vers l'extérieur de $Conv(\mathcal{F}_{\text{PLNE}})$. Il s'agit alors de couper toute la direction irréalisable du cône des directions réalisables en $\boldsymbol{x}^k$. Pour couper toute la direction, à partir de $\boldsymbol{x}^k$, il faut donc que l'hyperplan de séparation soit saturé en $\boldsymbol{x}^k$, c'est-à-dire que $\boldsymbol{\alpha}^T\boldsymbol{x}^k = \beta$. Ainsi, le *problème de séparation primale* se formule de façon légèrement différente du problème de séparation standard **SEP**.

P-SEP	Étant donné une solution réalisable $\boldsymbol{x}^k \in \mathcal{F}_{\text{PLNE}}$ et un point non réalisable $\boldsymbol{x}^\star \notin \mathcal{F}_{\text{PLNE}}$, déterminer un hyperplan qui sépare $\boldsymbol{x}^\star$ de $\mathcal{F}_{\text{PLNE}}$ et saturé au point $\boldsymbol{x}^k$ ou prouver qu'il n'en existe pas.

De tels hyperplans peuvent ensuite être ajoutés à **fAUG** pour améliorer sa qualité en tant que relaxation de **iAUG** : les directions non réalisables pour $Conv(PLNE)$ ne présentent aucun intérêt si l'on cherche une meilleure solution entière, donc en supprimer de la relaxation ne peut qu'améliorer celle-ci. On appelle *inégalité valide primale pour* $\boldsymbol{x}^k$ toute inégalité valide pour $\mathcal{F}_{\text{PLNE}}$ qui est saturée en $\boldsymbol{x}^k$, et *plan coupant primal* (ou *coupe primale*) *pour* $\boldsymbol{x}^k$ *et* $\boldsymbol{x}^\star$ toute inégalité valide primale saturée en $\boldsymbol{x}^k$ qui sépare $\boldsymbol{x}^\star$ de $\mathcal{F}_{\text{PLNE}}$. Quand le contexte est clair, on ne précise pas à quel(s) point(s) ces inégalités et coupes se rattachent. Un exemple de séparation primale est donné sur la figure 2.1.

En même temps que naissaient les premières méthodes primales, apparaissaient les premiers algorithmes de séparation de plans coupants primaux. Gomory (1958) décrit une famille de coupes primales qui permettent de résoudre PLNE à l'optimalité de façon autonome, sans nécéssiter de branchement ou d'autre technique pour assurer de trouver l'optimum entier. Une grande partie des travaux récents concernant les algorithmes primaux traitent en fait du problème de séparation primale **P-SEP**. Ainsi, Eisenbrand et al. (2003) prouvent que **P-SEP** est polynomialement équivalent (en termes de temps de résolution) au problème

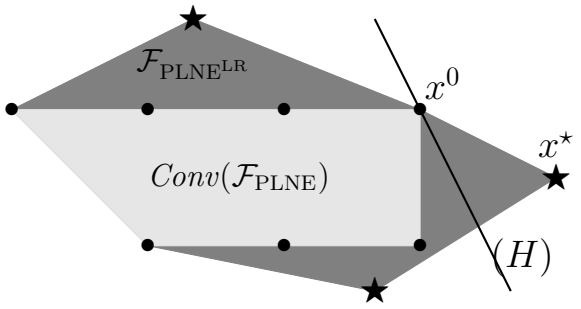


Figure 2.1 Exemple de séparation primale. (H) est une coupe primale saturée en $\boldsymbol{x}^0$ et séparant $\boldsymbol{x}^\star$ de $\mathcal{F}_{\text{PLNE}}$. Le domaine de la relaxation linéaire $\mathcal{F}_{\text{PLNE}^{\text{LR}}}$ est le polyèdre gris foncé. Le domaine de PLNE est un ensemble fini représenté par les points (•), et son enveloppe convexe $Conv(\mathcal{F}_{\text{PLNE}})$ est le polyèdre gris clair.

d'optimisation PLNE. De même que pour **iAUG**, il ne faut donc pas s'attendre à trouver de méthodes de séparation générique en temps polynomial. Il existe toutefois des travaux proposant des algorithmes de séparation pour des familles de coupes primales spécifiques, généralement non exhaustives, c'est-à-dire qu'elles ne suffisent pas à décrire $Conv(\mathcal{F}_{\text{PLNE}})$ et leur seule utilisation ne permet pas de résoudre le problème d'optimisation.

Firla et al. (1999) exposent un parallèle intéressant entre les algorithmes de plans coupants primaux et standards, en insistant en particulier sur le problème de b-matching, sans pour autant présenter de résultats numériques. Letchford & Lodi (2002, 2003b) passent en revue plusieurs algorithmes de séparation primale dont la plupart sont l'adaptation au cas primal de familles de coupes standard efficaces en pratique. Les mêmes auteurs proposent ensuite un cadre générique pour un algorithme primal dans lequel les coupes jouent un rôle prépondérant (Letchford & Lodi, 2003a). Des résultats numériques sont présentés pour de petits problèmes de sac-à-dos multidimensionnel. Bien que leur travail ne mentionne pas explicitement les méthodes primales, Ralphs & Galati (2006) développent une méthode de plans coupants adaptée à la décomposition de type Dantzig-Wolfe, dont le sous-problème peut être rapproché du problème d'augmentation. La méthode qu'ils proposent présente des similitudes avec la recherche de coupes primales et leur travail mérite d'être mentionné ici. Ils s'intéressent à plusieurs problèmes particuliers dont, entre autres, le b-matching et les tournées de véhicules.

2.5.3 Cas particulier du simplexe en nombres entiers

Comme nous l'allons voir, dans le cas des problèmes binaires, et particulièrement dans celui du problème de partitionnement, il est possible de concevoir des algorithmes primaux dont l'obtention de solutions successives est basée sur l'exécution de pivots de simplexe.

Fondements théoriques et intérêt pratique

Plaçons-nous d'abord dans le cadre général de la programmation linéaire en nombres entiers. Soit $\boldsymbol{x}^0 \in \mathcal{F}_{\text{PLNE}}$. Comme $Conv(\mathcal{F}_{\text{PLNE}})$ est un polytope, il existe une suite $\left(\boldsymbol{x}^k\right)_{0\leq k\leq K}$ de points de $Ext\left(Conv(\mathcal{F}_{\text{PLNE}})\right)$ de coût décroissant, telle que pour tout $k \in \{1,\ldots,K\}$, $\boldsymbol{x}^k$ est voisin de $\boldsymbol{x}^{k-1}$ dans $Conv(\mathcal{F}_{\text{PLNE}})$, et $\boldsymbol{x}^K$ est une solution optimale de $PLNE$. Un algorithme capable de déterminer une telle suite, à partir de la solution initiale, est appelé un simplexe en nombres entiers (*integral simplex*). De façon équivalente, il s'agit d'un algorithme primal (`C1`–`C3`) qui vérifie

`C4` $\boldsymbol{x}^{k+1}$ est un voisin de $\boldsymbol{x}^k$ dans $Conv(\mathcal{F}_{\text{PLNE}})$.

Plaçons-nous désormais dans le cas de SPP. Soit $\left(\boldsymbol{x}^k\right)_{0\leq k\leq K}$ la suite de solutions fournie par un simplexe en nombre entiers et $k \in \{1,\ldots,K\}$. SPP est un programme binaire, donc $\boldsymbol{x}^k \in Ext\left(\mathcal{F}_{\text{SPP}^{\text{LR}}}\right)$. De plus, SPP est quasi-intégral, donc $\boldsymbol{x}^k$ est voisin de $\boldsymbol{x}^{k-1}$ dans $\mathcal{F}_{\text{SPP}^{\text{LR}}}$ et on peut ainsi passer de $\boldsymbol{x}^{k-1}$ à $\boldsymbol{x}^k$ en effectuant un ou plusieurs pivots de simplexe tel que procède un algorithme de résolution de la relaxation linéaire. Ainsi, il est possible de concevoir un algorithme effectuant des séries de pivots de simplexe qui, à partir d'une solution entière, fournit une suite de solutions entières de coût décroissant pour le problème de partitionnement. Dans le cas d'un simplexe en nombres entiers, toutes les séries de pivots ne sont pas autorisées, mais seulement celles qui permettent d'aller d'un sommet entier à un autre sommet entier de $\mathcal{F}_{\text{SPP}^{\text{LR}}}$. La difficulté ne provient pas du caractère décroissant de l'arête à suivre (le simplexe trouve très efficacement de telles arêtes), mais bien du fait que toutes les arêtes décroissantes ne mènent pas vers un voisin entier.

En termes d'applications pratiques, les algorithmes de type primaux présentent de nombreux avantages. Premièrement, capables de tirer parti de solutions initiales connues, ils sont parfaitement adaptés à la réoptimisation. Par exemple, dans le cas de la planification de personnel navigant dans le domaine aérien, l'emploi du temps déterminé à l'avance doit souvent être modifié en réponse à des contraintes opérationnelles telles que les retards, les substitutions d'aéronefs, les absences, les grèves ou les éruptions volcaniques (Stojković et al., 1998). Deuxièmement, considérons le cas de la génération de colonnes purement entière (*all-integer column generation*). La génération de colonnes est une technique d'op-

timisation introduite pour la première fois par Dantzig & Wolfe (1960) et utilisée dans le cadre de la résolution de problèmes dont le nombre de variables est très grand.

Dans cette méthode, on ne considère au départ qu'un sous-ensemble des variables, et donc des colonnes de $\boldsymbol{A}$. La restriction du problème original aux variables sélectionnées forme le *problème maître restreint* (*restricted master problem*, RMP) ; ce problème est résolu à l'optimalité. Il est important de sélectionner suffisament de ces variables au départ, de telle façon à ce que le domaine de RMP ne soit pas vide. Étant donnée une solution optimale de RMP, on formule un *sous-problème* dont la résolution permet de sélectionner (ou *générer*) des variables de coût réduit minimal parmi celles qui ne sont pas déjà dans RMP. Si ce dernier est positif (cas d'une minimisation), la solution courante est optimale pour le problème global ; sinon, il suffit d'ajouter une ou plusieurs variables de coût réduit négatif à RMP pour en améliorer la solution optimale. On itère le processus jusqu'à atteindre l'optimalité. Tous les problèmes ne se prêtent pas à l'application de ce type de schéma, mais seulement ceux pour lesquels ils est possible de formuler et résoudre efficacement le sous-problème.

Dans le cas où les variables du problème doivent prendre des valeurs entières, la génération de colonnes est traditionnellement intégrée à un schéma d'énumération implicite : de nouvelles colonnes peuvent être ajoutées à chaque nœud de l'arbre de branchement. Bien que les problèmes résolus ne soient alors que des relaxations linéaires, le parcours de l'arbre de branchement peut se révéler extrêmement long car sa taille explose rapidement. Plutôt que d'être insérée dans un *branch-and-bound*, la génération de colonnes peut être intégrée à un algorithme primal. Dans ce cas, étant donné la solution optimale (entière) du problème maître à l'étape k, de nouvelles colonnes de coût réduit négatif sont ajoutées, puis le nouveau problème (étape $k+1$) est résolu à l'optimalité (toujours en nombres entiers). Pour accélérer ce processus, il semble impératif d'utiliser la solution optimale du problème à l'étape k comme point de départ pour la résolution du problème étendu de l'étape $k+1$. Ainsi que nous l'avons expliqué à la section 2.4.2, les méthodes adaptées à ces cas de figures sont les méthodes primales ; d'où la nécessité d'un algorithme de réoptimisation performant si l'on souhaite développer une procédure de génération de colonnes purement entière.

État de l'art

Au regard (1) de la nature intrinsèquement dégénérée du problème de partitionnement, particulièrement dans les applications industrielles, et (2) des problèmes causés par la dégénerescence aux algorithmes primaux, spécifiquement à ceux basés sur des pivots de simplexe, et lors de l'ajout de plans coupants (voir Letchford & Lodi, 2003b), il paraît essentiel d'intégrer des techniques visant à pallier la dégénerescence dans les implémentations de simplexes en

nombres entiers. Un des premiers algorithmes à prendre en compte ce phénomène est celui de Balas & Padberg (1975). Étant donné une solution $\boldsymbol{x}^k \in \mathcal{F}_{\text{SPP}}$, un ensemble important de directions potentielles est généré, puis réduit aux seules directions des arêtes menant vers les voisins entiers de $\boldsymbol{x}^k$ au moyen d'éliminations successives. Pour tout voisin entier $\boldsymbol{x}'$ de $\boldsymbol{x}^k$, la direction correspondante est $\boldsymbol{d} = \boldsymbol{x}' - \boldsymbol{x}^k$. Le support positif de $\boldsymbol{d}$, $Q^+ = \{j \mid d_j > 0\}$ définit alors l'ensemble des colonnes à pivoter en base pour passer de $\boldsymbol{x}^k$ à $\boldsymbol{x}'$. Dans les termes de Balas & Padberg, Q^+ est un ensemble non décomposable si, pour tout sous-ensemble strict $\tilde{Q}^+ \subset Q^+$, effectuer les pivots sur les variables d'indices $\tilde{Q}^+$ ne permet pas de changer de solution. Balas & Padberg prouvent que l'ensemble Q^+ est non décomposable si et seulement si $\boldsymbol{x}^k$ et $\boldsymbol{x}'$ sont voisins dans $Conv(\mathcal{F}_{\text{SPP}})$ ou, de façon équivalente, dans $\mathcal{F}_{\text{SPP}^{\text{LR}}}$. Une fois toutes ces directions déterminées, il est alors facile de choisir celle de plus grande pente, ou en d'autres termes de plus faible coût réduit, puis d'effectuer la série de pivots correspondant à cette direction. Le résultat de cette opération est indépendant de l'ordre dans lequel les variables sont pivotées. À chaque étape, si $\boldsymbol{x}^k$ n'est pas optimal pour SPP, la nouvelle solution est voisine de $\boldsymbol{x}^k$, et de coût strictement inférieur. Le défaut de cette méthode provient de l'explosion du nombre de voisins entiers de $\boldsymbol{x}^k$ quand la dimension n de l'espace augmente, et du très grand nombre de directions potentielles qu'il faut éliminer pour se restreindre aux seules directions entières.

Thompson (2002) introduit le concept de méthode de simplexe entier (*Integral Simplex Method*, ISM). Cette méthode se divise en deux niveaux d'optimisation dont le premier est le simplexe en nombres entiers local (*Local Integral Simplex Method*, LISM), et le second le simplexe en nombre entier global (*Global Integral Simplex Method*, GLISM) Le LISM repose sur une succession de *pivots-sur-1*, c'est-à-dire des pivots sur des entrées du tableau de simplexe de valeur 1, et se limite à des pivots améliorants. Dans le cas de SPP, les pivots-sur-1 sont les pivots de simplexe permettant de passer d'une solution entière à une autre. La restriction à ce type de pivots ne permet cependant pas d'atteindre l'optimalité et Thompson insère le LISM dans le cadre plus global du GLISM qui est un schéma d'arbre de branchement. Dans GLISM, à chaque nœud de l'arbre de branchement, un sous-problème est résolu par LISM. Si la solution trouvée est optimale pour le sous-problème, l'exploration de la branche s'arrête ; sinon, de nouvelles sous-branches sont créées, et ainsi de suite jusqu'à exploration complète de l'arbre. Thompson montre des résultats numériques prometteurs pour cette méthode, sur des instances classiques de la littérature (instances de Hoffman & Padberg, 1993). Se limiter aux seuls pivots-sur-1 est somme toute assez restrictif et la taille de l'arbre de branchement parcouru par GLISM pour compenser cette restriction peut exploser. Des méthodes de propagation de contraintes sont proposées pour accélérer la résolution des programmes linéaires si des décisions de branchement ont été prises, mais la taille de l'arbre de branchement reste

exponentielle. Thompson borne le nombre de pivots à effectuer par $2(n^2+n)^m$. Malheureusement, dans de nombreuses applications pratiques, m tend à être de plus en plus grand. Dans le cas de la création d'emploi du temps par exemple, m représente le nombre de tâches à effectuer. Dans les problèmes de planification de personel navigant sur une semaine et des horaires de chauffeurs de bus sur une journée, il n'est pas rare que m atteigne des valeurs supérieures à $1,000$. Saxena (2003a,b) ajoute des extensions essentiellement théoriques aux travaux de Thompson : caractérisation des solutions voisines et des bases correspondantes, gestion du phénomène de cycle dans le simplexe et ajout de plans coupants.

Stallmann & Brglez (2007) présentent une étude comparative de leur implémentation d'un algorithme dual en nombres entiers pour des problèmes de couverture d'ensemble (SPP dont les contraintes d'égalités sont remplacées par des inégalités $\geq$). Bien qu'ils ne proposent pas d'avancée théorique, leur travail est un excellent exemple d'étude comparative d'un algorithme purement entier avec le solveur générique de CPLEX. Il s'agit aussi d'un des rares articles à fournir des résultats numériques sur des problèmes de taille conséquente.

Parmi les travaux contemporains concernant le simplexe en nombres entiers, un des plus intéressants est celui mené par Rönnberg dans le cadre de sa thèse de doctorat (Rönnberg, 2012). D'une part, le simplexe en nombres entiers y est implémenté comme sous-routine d'un processus de génération de colonnes purement entière (Rönnberg & Larsson, 2009), d'autre part, des résultats sont présentés dans le cas d'une application à la planification d'emploi du temps d'infirmières (Rönnberg & Larsson, 2014). Cependant, l'algorithme développé autorise non seulement les pivots-sur-1, mais aussi les pivots-sur-(-1) qui sont dégénérés et ne permettent donc pas de changer de sommet du polyèdre. Cette caractéristique théorique rend l'algorithme vulnérable au phénomène de dégénerescence.

Enfin, le travail qui nous intéresse le plus ici est celui de Zaghrouti et al. (2014) : le simplexe en nombres entiers avec décomposition (*Integral Simplex Using Decomposition*, ISUD). Cet algorithme est le fruit de l'adaptation au cas entier de travaux antérieurs sur des méthodes tournant la dégénérescence en un atout pour le simplexe en programmation linéaire (Elhallaoui et al., 2005, 2010, 2011; Metrane et al., 2010; Omer et al., 2015). Ainsi que nous l'avons déjà évoqué, il semble impératif pour un algorithme basé sur des pivots de simplexe de pallier les effets de la dégénérescence. C'est la raison pour laquelle ISUD est probablement le meilleur candidat pour être l'algorithme primal adapté au problème de partitionnement – et ultimement à d'autres problèmes binaires, voir entiers. ISUD repose sur la division des indices des variables en trois sous-ensembles $\mathcal{P}$, $\mathcal{C}$ et $\mathcal{I}$ que sont respectivement la *base de travail*, les indices des variables *compatibles* et ceux des variables *incompatibles*. Étant donné une solution courante $\boldsymbol{x}^k$, la base de travail $\mathcal{P} = \left\{ j \mid x_j^k = 1 \right\}$ décrit les indices des variables

prenant des valeurs non nulles en $\boldsymbol{x}^k$. L'indice $j \in \{1,\ldots,n\} \setminus \mathcal{P}$ d'une variable pour laquelle $x_j^k = 0$ est compatible si la colonne correspondante $\boldsymbol{A}_{\cdot j}$ est dans l'espace vectoriel engendré par les colonnes de la solution courante $\mathsf{Span}\,(\boldsymbol{A}_{\cdot P})$; et incompatible dans le cas inverse. Cette partition des variables permet de décomposer le problème d'augmentation **iAUG** ainsi que sa relaxation linéaire et donc d'accélérer sa résolution. La décomposition du problème selon cette partition n'implique pas de perte d'information, ni n'empêche de prouver l'optimalité. Le présent travail de doctorat se base essentiellement sur cet algorithme qui sera présenté en détails aux chapitres 4 à 6.

2.6 Simplexe en nombres entiers avec décomposition

La description de ISUD donnée ici se base d'une part sur le travail de Zaghrouti et al., ainsi que d'autre part sur ceux de Rosat et al. (2014, 2015a) et d'Omer et al. (2015). Une des contributions apportées dès le début de ce projet de recherche fut de reformuler, réécrire, et améliorer la présentation de l'algorithme, ainsi que de proposer une étude théorique plus poussée du fonctionnement de ISUD. Ces avancées ont permis de le rapprocher des algorithmes primaux, en ne mentionnant plus la théorie de la dualité dans la description de la méthode. La forme de la présentation de l'algorithme ainsi que certains des résultats de la fin de ce chapitre sont donc des contributions originales de cette thèse. Pour rendre le texte de cette section plus clair, aucune démonstration n'est donnée ; celles-ci se trouvent dans les références citées ci-avant.

On suppose dans toute cette section qu'une solution réalisable de SPP, $\boldsymbol{x}^k \in \mathcal{F}_{\text{SPP}}$, est connue. Comme pour tout programme linéaire binaire, $\boldsymbol{x}^k$ est un point extrême de $\mathcal{F}_{\text{SPP}^{\text{LR}}}$ et donc une solution de base. Soit $\mathcal{P} = \left\{ j \in \{1,\ldots,n\} \mid x_j^k = 1 \right\}$ l'ensemble des indices des variables non nulles en $\boldsymbol{x}^k$ (*base de travail*), et $\mathcal{Z} = \{1,\ldots,n\} \setminus \mathcal{P} = \left\{ j \in \{1,\ldots,n\} \mid x_j^k = 0 \right\}$ l'ensemble des indices des autres variables (nulles en $\boldsymbol{x}^k$). Le problème à résoudre est **iAUG** : trouver une solution $\boldsymbol{x}^{k+1} \in \mathcal{F}_{\text{SPP}}$ de coût strictement moindre que $\boldsymbol{x}^k$ ou prouver que $\boldsymbol{x}^k$ est optimale pour SPP. Pour ce faire, l'algorithme consiste à formuler une relaxation linéaire du problème d'augmentation, après avoir décomposé l'espace de recherche des solutions en deux sous-espaces plus petits.

2.6.1 Augmentation fractionnaire fAUG

Avant d'entreprendre la résolution du problème d'augmentation entier tel que formulé ci-avant, considérons le problème d'augmentation *fractionnaire* suivant, pour lequel la nouvelle solution $\boldsymbol{x}^{k+1}$ peut n'être qu'une solution de SPP^{LR}. Des définitions utiles pour la suite sont

données après l'explicitation de **fAUG**.

fAUG	Déterminer un vecteur d'augmentation $\boldsymbol{z} \in \mathbb{R}^n$ tel que $\left(\boldsymbol{x}^k + \boldsymbol{z}\right) \in \mathcal{F}_{\text{SPP}^{\text{LR}}}$ et $\boldsymbol{c}^T\boldsymbol{z} < 0$.

Définition 2. Soit $\boldsymbol{d} \in \mathbb{R}^n$. $\boldsymbol{d}$ est une *direction réalisable* en $\boldsymbol{x}^k$ s'il existe un pas $\rho > 0$ tel que $\left(\boldsymbol{x}^k + \rho\boldsymbol{d}\right) \in \mathcal{F}_{\text{SPP}^{\text{LR}}}$; dans ce cas on définit le plus long pas associé à cette direction comme $r(\boldsymbol{d}) = \max\left\{\rho > 0 \mid \left(\boldsymbol{x}^k + \rho\boldsymbol{d}\right) \in \mathcal{F}_{\text{SPP}^{\text{LR}}}\right\}$. De plus, si $\left(\boldsymbol{x}^k + r(\boldsymbol{d})\boldsymbol{d}\right)$ est une solution entière, on dit que $\boldsymbol{d}$ est une *direction entière*. Enfin, si $\boldsymbol{c}^T\boldsymbol{d} < 0$, $\boldsymbol{d}$ est une direction d'*augmentation*.

Un vecteur $\boldsymbol{z}$ solution de **fAUG** peut donc être défini comme le produit $\boldsymbol{z} = r\boldsymbol{d}$ d'un pas r et d'une direction d'augmentation réalisable $\boldsymbol{d}$. L'ensemble des directions réalisables en $\boldsymbol{x}^k$ est le cône

$$\Gamma = \{\boldsymbol{d} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{d} = 0, \boldsymbol{d}_{\mathcal{P}} \leq \boldsymbol{0}, \boldsymbol{d}_{\mathcal{Z}} \geq 0\}, \tag{2.1}$$

dont on ne considérera que la section

$$\Delta = \Gamma \cap \left\{\boldsymbol{d} \in \mathbb{R}^n \mid \boldsymbol{e}^T\boldsymbol{d}_{\mathcal{Z}} = 1\right\}. \tag{2.2}$$

La contrainte linéaire $\boldsymbol{e}^T\boldsymbol{d}_{\mathcal{Z}} = 1$ est appelée contrainte de *normalisation*. Remarquons ici que, comme tous les termes de $\boldsymbol{A}$ sont non négatifs, toute direction d'amélioration non nulle $\boldsymbol{d} \in \Gamma$ possède au moins un élément non nul d_j pour un certain $j \in \mathcal{Z}$, donc Δ est bien une section du cône Γ. Il est plus simple de chercher une direction d'amélioration dans le polytope Δ que dans le cône Γ, car c'est un domaine borné. On note Δ^{int} l'ensemble des directions entières de Δ (celles qui mènent vers un autre point entier du polyèdre $\mathcal{F}_{\text{SPP}^{\text{LR}}}$). On a $\Delta^{int} = \left\{\boldsymbol{d} \in \Delta | \boldsymbol{x}^k + r(\boldsymbol{d})\boldsymbol{d} \in \mathcal{F}_{\text{SPP}}\right\}$.

Il existe une direction d'amélioration en $\boldsymbol{x}^k$ si et seulement si le problème suivant

$$z^\star_{\text{MIMA}} = \min_{\boldsymbol{d} \in \mathbb{R}^n} \left\{\boldsymbol{c}^T\boldsymbol{d} \mid \boldsymbol{d} \in \Delta\right\}, \tag{MIMA}$$

appelé problème de l'augmentation maximale en moyenne entrante (*Maximum Incoming-Mean Augmentation*), vérifie $z^\star_{\text{MIMA}} < 0$. Dans ce cas, étant donné $\boldsymbol{d}^\star \in \mathcal{F}_{\text{MIMA}}$ une de ses solutions optimales (direction réalisable d'augmentation), l'idée est de suivre cette direction aussi loin que les contraintes linéaires le permettent en choisissant $\boldsymbol{z} = r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$ comme vecteur d'amélioration. La nouvelle solution, potentiellement fractionnaire, est donc $\boldsymbol{x}^\star = \boldsymbol{x}^k + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$.

2.6.2 Décomposition

Nous présentons ici une partition de l'ensemble des variables qui permet de décomposer le problème d'augmentation.

Définition 3. Soit $j \in \mathcal{Z}$ l'indice d'une variable hors base, et $\boldsymbol{A}_{\cdot j}$ la colonne de $\boldsymbol{A}$ correspondante. $\boldsymbol{A}_{\cdot j}$ est dite *compatible* si elle appartient à l'espace vectoriel engendré par les colonnes de la base de travail $\mathsf{Span}\left(\boldsymbol{A}_{\cdot \mathcal{P}}\right)$; sinon on dit qu'elle est *incompatible*. Par extension, cette dénomination s'applique aux variables et aux indices correspondants. L'ensemble des indices des variables compatibles est noté $\mathcal{C}$, celui des indices des variables incompatibles $\mathcal{I}$.

Les ensembles $(\mathcal{P}, \mathcal{C}, \mathcal{I})$ forment une partition de $\{1, \ldots, n\}$. Sans perte de généralité, à une permutation des lignes et des colonnes de $\boldsymbol{A}$ près, on peut supposer $\mathcal{P} = \{1, \ldots, p\}$. En notant $\bar{\mathcal{P}} = \{1, \ldots, m\} \setminus \mathcal{P}$, on peut alors écrire la matrice sous la forme :

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I}_p & \boldsymbol{A}_{\mathcal{P}\mathcal{C}} & \boldsymbol{A}_{\mathcal{P}\mathcal{I}} \\ \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}} & \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{C}} & \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{I}} \end{bmatrix}, \tag{2.3}$$

où $\boldsymbol{I}_p$ désigne la matrice identité de rang p. Avec ces notations, on peut introduire les deux sous-problèmes qui nous intéressent, respectivement appelés Mima restreint (*Restricted* Mima) et Mima complémentaire (*Complementary* Mima) :

$$z^\star_{\text{R-Mima}} = \min_{\boldsymbol{d} \in \mathbb{R}^{p+|\mathcal{C}|}} \left\{ \boldsymbol{c}_\mathcal{P}^T \boldsymbol{d}_\mathcal{P} + \boldsymbol{c}_\mathcal{C}^T \boldsymbol{d}_\mathcal{C} \,|\, \boldsymbol{A}_{\cdot\mathcal{P}} \boldsymbol{d}_\mathcal{P} + \boldsymbol{A}_{\cdot\mathcal{C}} \boldsymbol{d}_\mathcal{C} = 0\,,\, \boldsymbol{e}_\mathcal{C}^T \boldsymbol{d}_\mathcal{C} = 1\,,\, \boldsymbol{d}_\mathcal{C} \geq 0 \right\} \quad \text{(R-Mima)}$$

$$z^\star_{\text{C-Mima}} = \min_{\boldsymbol{d} \in \mathbb{R}^{p+|\mathcal{I}|}} \left\{ \boldsymbol{c}_\mathcal{P}^T \boldsymbol{d}_\mathcal{P} + \boldsymbol{c}_\mathcal{I}^T \boldsymbol{d}_\mathcal{I} \,|\, \boldsymbol{A}_{\cdot\mathcal{P}} \boldsymbol{d}_\mathcal{P} + \boldsymbol{A}_{\cdot\mathcal{I}} \boldsymbol{d}_\mathcal{I} = 0\,,\, \boldsymbol{e}_\mathcal{I}^T \boldsymbol{d}_\mathcal{I} = 1\,,\, \boldsymbol{d}_\mathcal{I} \geq 0 \right\} \quad \text{(C-Mima)}$$

Dans le cas continu, pour l'algorithme IPS, Elhallaoui et al. (2011) montrent que $z^\star_{\text{Mima}} < 0$ si et seulement si $z^\star_{\text{R-Mima}} < 0$ ou $z^\star_{\text{C-Mima}} < 0$. Ainsi, pour déterminer si une direction d'amélioration existe, il suffit de montrer que la valeur optimale de l'un ou l'autre de ces problèmes d'optimisation est strictement négative. On note respectivement $\Delta_\mathcal{C} = \{\boldsymbol{d} \in \Delta | \boldsymbol{d}_\mathcal{I} = \boldsymbol{0}\}$ et $\Delta_\mathcal{I} = \{\boldsymbol{d} \in \Delta | \boldsymbol{d}_\mathcal{C} = \boldsymbol{0}\}$ les ensembles des directions réalisables restreints à des variables entrantes compatibles et incompatibles. On a donc : $\mathcal{F}_{\text{R-Mima}} = \Delta_\mathcal{C}$ et $\mathcal{F}_{\text{C-Mima}} = \Delta_\mathcal{I}$. Omer et al. (2015) étendent ce résultat et montrent que

$$z^\star_{\text{Mima}} = \min\left\{ z^\star_{\text{R-Mima}}, z^\star_{\text{C-Mima}} \right\}, \tag{2.4}$$

justifiant ainsi pleinement la décomposition proposée.

2.6.3 Augmentation entière

Cherchons désormais à résoudre le problème d'augmentation entière **iAUG**. Zaghrouti et al. (2014) ont montré que le résultat d'Elhallaoui et al. (2011) reste valide dans le cas des solutions entières du problème de partitionnement : il existe une direction $\boldsymbol{d}$ d'amélioration entière si et seulement s'il en existe une pour laquelle $\boldsymbol{d}_{\mathcal{C}} = \mathbf{0}$ ou $\boldsymbol{d}_{\mathcal{I}} = \mathbf{0}$. Rosat et al. (2014) prouvent qu'étant donné $j \in \mathcal{Z}$, il existe exactement une solution binaire $\boldsymbol{x}' \in \mathcal{F}_{\text{SPP}}$ différente de $\boldsymbol{x}^k$, telle que les indices des variables non nulles de $\boldsymbol{x}'$ sont dans $\mathcal{P} \cup \{j\}$ si et seulement si $j \in \mathcal{C}$. Passer de $\boldsymbol{x}^k$ à cette nouvelle solution est équivalent à effectuer un pivot de simplexe, mais seulement sur la base de travail $\mathcal{P}$; par abus de langage, cette opération est aussi appelée *pivot*. Si $j \in \mathcal{I}$, l'unique solution réalisable de SPP^{LR} telle que ses seules variables non nulles sont dans $\mathcal{P} \cup \{j\}$ est $\boldsymbol{x}^k$. Ces observations conduisent à des méthodes de résolution différentes suivant que l'on cherche une direction d'amélioration entière dont le support est compatible (dans $\mathcal{P} \cup \mathcal{C}$) ou incompatible (dans $\mathcal{P} \cup \mathcal{I}$). On restreint alors **iAUG** à $\mathcal{C}$ d'une part, et à $\mathcal{I}$ d'autre part. De plus, Rosat et al. (2015a) prouvent le pendant de l'équation 2.4 pour le cas entier : le coût de la meilleure direction entière est égal au minimum du coût de la meilleure solution entière compatible, ou à celui de la meilleure direction entière incompatible.

Augmentation entière compatible

Dans la mesure où l'insertion de toute colonne compatible dans la base de travail fournit une nouvelle solution entière voisine de $\boldsymbol{x}^k$, il suffit de déterminer l'indice $j \in \mathcal{C}$ d'une variable de $\mathcal{C}$ de coût réduit minimal pour établir s'il existe un pivot non dégénéré améliorant. En définissant le vecteur des coûts réduits des variables hors-base comme $\bar{\boldsymbol{c}}_{\mathcal{Z}}^T = \boldsymbol{c}_{\mathcal{Z}}^T - \boldsymbol{c}_{\mathcal{P}}^T \boldsymbol{A}_{\mathcal{P}\mathcal{Z}}$, il s'agit en fait de déterminer si le problème suivant

$$z_{\text{RP}}^\star = \min_{j \in \mathcal{C}} \{\bar{c}_j\}\,, \tag{RP}$$

appelé problème réduit (*Reduced Problem*), vérifie $z_{\text{RP}}^\star < 0$. Si c'est le cas, il suffit de pivoter une variable x_j en base de coût réduit $\bar{c}_j$ strictement négatif pour obtenir une solution entière, de coût strictement inférieur à celui de $\boldsymbol{x}^k$, et voisine de $\boldsymbol{x}^k$ dans $Conv(\mathcal{F}_{\text{SPP}})$; sinon, il faut s'attaquer à la restriction de **iAUG** à $\mathcal{I}$.

Augmentation entière incompatible

On note $\Delta_{\mathcal{I}}^{int} = \Delta_{\mathcal{I}} \cap \Delta^{int}$ le sous-ensemble des directions entières de $\Delta_{\mathcal{I}}$. Pour déterminer une augmentation entière dont les variables entrantes en base sont incompatibles, il faut

identifier une direction $\boldsymbol{d} \in \Delta_{\mathcal{I}}^{int}$ de coût $\bar{\boldsymbol{c}}^T\boldsymbol{d}$ strictement négatif. C'est-à-dire résoudre le programme non linéaire – l'ensemble $\Delta_{\mathcal{I}}^{int}$ est discret – suivant :

$$\min\left\{\boldsymbol{c}^T\boldsymbol{d} \,|\, \boldsymbol{d} \in \Delta_{\mathcal{I}}^{int}\right\}. \tag{iC-Mima}$$

Comme $\Delta_{\mathcal{I}}^{int} \subseteq \Delta_{\mathcal{I}}$, C-Mima est une relaxation de iC-Mima. Et comme c'est un programme linéaire, il s'agit d'une relaxation linéaire. Rappelons que tout programme linéaire dont le domaine est borné admet une solution optimale qui est un sommet de ce polyèdre (propriété énoncée à la section 2.3.1). Ainsi, si l'on considère le programme linéaire $\min\left\{\boldsymbol{c}^T\boldsymbol{d} \,|\, \boldsymbol{d} \in \mathit{Conv}(\Delta_{\mathcal{I}}^{int})\right\}$, on peut déduire qu'au moins une des solutions optimales de iC-Mima est un point extrême de $\Delta_{\mathcal{I}}^{int}$ ($\Delta_{\mathcal{I}}^{int}$ est un ensemble fini, donc ses points extrêmes coïncident avec ceux de son enveloppe convexe).

Une analyse succinte de la structure polyédrale des domaines des différents programmes linéaires introduits ci-avant permet de déduire les propriétés suivantes (Rosat et al., 2015a). Premièrement, $\mathit{Ext}\,(\Delta)$ est exactement l'ensemble des directions des arêtes du polyèdre $\mathcal{F}_{\text{SPP}^{\text{LR}}}$ passant par $\boldsymbol{x}^k$. Deuxièmement, comme SPP est quasi-intégral, les points extrêmes de l'ensemble des directions réalisables entières sont aussi des directions extrêmes de l'ensemble des directions réalisables, c'est-à-dire $\mathit{Ext}\,(\Delta^{int}) \subseteq \mathit{Ext}\,(\Delta)$. Ainsi, la relaxation linéaire C-Mima possède l'avantage suivant : au moins une solution optimale de iC-Mima se trouve parmi l'ensemble des points extrêmes de C-Mima.

Définition 4. Soit $\mathcal{J} \subseteq \{1, \ldots, n\}$. L'ensemble des colonnes de $\boldsymbol{A}$ d'indices dans $\mathcal{J}$ est *colonnes-disjoint* si aucune paire de colonnes de $\boldsymbol{A}_{\cdot\mathcal{J}}$ n'a d'entrée non nulle sur la même ligne. Comme $\boldsymbol{A}$ est binaire, il est équivalent de dire que pour tous $j_1, j_2 \in \mathcal{J}$, $j_1 \neq j_2$, on a $\boldsymbol{A}_{\cdot j_1}^T \boldsymbol{A}_{\cdot j_2} = 0$. Par extension, on dit de $\mathcal{J}$ et $\boldsymbol{A}_{\cdot\mathcal{J}}$ qu'ils sont colonnes-disjoints.

Zaghrouti et al. (2014) ont montré qu'un point extrême $\boldsymbol{d} \in \mathit{Ext}\,(\Delta_{\mathcal{I}})$ est une direction entière si et seulement si $\mathit{Supp}(\boldsymbol{d}_{\mathcal{I}})$ est colonnes-disjoint. On dispose donc d'une caractérisation simple des directions extrêmes entières. Comme la plupart des algorithmes pour la programmation linéaire fournissent des solutions qui sont des points extrêmes du polyèdre des contraintes, cette caractérisation s'avère très utile. Avant le commencement de ce projet de thèse, si la solution optimale de la relaxation C-Mima n'était pas entière, la seule méthode proposée pour l'« améliorer » était de fixer toutes les variables du support de $\boldsymbol{d}_{\mathcal{I}}^{\star}$ à zéro dans C-Mima et de la résoudre de nouveau (branchement non exhaustif). Lorsque ce processus permet de trouver une direction entière $\boldsymbol{d}^{\star}$ de coût strictement négatif, il est alors relancé depuis la nouvelle solution $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + r(\boldsymbol{d}^{\star})\boldsymbol{d}^{\star}$; cependant, quand un nombre trop important de variables sont fixées à zéro et que C-Mima devient irréalisable, le processus était arrêté et la solution courante $\boldsymbol{x}^k$ retournée, bien qu'éventuellement non optimale.

Enfin, il est important de noter que la présentation de l'algorithme est ici partiellement incomplète. En effet, le nombre de lignes des problèmes R-Mima et C-Mima peut être réduits en utilisant des techniques d'agrégation de contraintes. Ces processus sont essentiels pour réduire le temps d'exécution de l'algorithme, mais il n'est pas réellement nécessaire de les exposer pour saisir les tenants et aboutissants du sujet de cette thèse. Pour cette raison, et pour ne pas surcharger cette partie du manuscrit, ils ne sont détaillés que dans les chapitres ultérieurs.

Pseudo-code de ISUD

Algorithme 1: Simplexe en nombres entiers avec décomposition (ISUD)

Entrées: $\boldsymbol{x}^0$, une solution réalisable de SPP.
Sorties: $\boldsymbol{x}^k$, une solution de SPP, potentiellement meilleure que $\boldsymbol{x}^0$.

```
1  k ← 0 ;
2  tant que vrai faire
3      Si nécessaire, mettre à jour les ensembles P, C, et I associés à x^k ;
4      si z*_RP < 0 alors
5          c̄_i ← une solution optimale de RP (i ∈ C) ;
6          x^{k+1} ← solution de base de SPP obtenue en pivotant x_i dans P;
7          k ← k + 1 ;
8      sinon
9          continue ← vrai ;
10         tant que continue = vrai faire
11             Si nécessaire, mettre à jour les ensembles P, C, et I associés à x^k ;
12             Si nécessaire, mettre à jour C-Mima ;
13             si z*_C-Mima < 0 alors
14                 d* ← une solution optimale de C-Mima ;
15                 si d*_I est colonnes-disjoint alors
16                     x^{k+1} ← x^k + r(d*)d* ; k ← k + 1 ; continue ← faux ;
17                 sinon
18                     si un certain critère est vérifié alors
19                         retourner x^k ;
20                     sinon
                           — Ajouter des plans coupants à C-Mima ;
                           — ou appliquer une stratégie de branchement ;
21             sinon
22                 retourner x^k;
```

CHAPITRE 3 ORGANISATION DE LA THÈSE

Cette thèse est consacrée à la recherche de solutions du problème d'augmentation menant vers une solution entière, adjacente de la solution courante dans le polyèdre de la relaxation linéaire du problème de partitionnement. Au chapitre précédent, nous avons d'abord décrit l'état de l'art concernant les algorithmes primaux puis détaillé l'algorithme qui nous intéressera tout au long de ce travail, ISUD. Forts de ces connaissances, nous proposons trois approches interdépendantes basées sur la reformulation du problème d'augmentation et l'amélioration de sa relaxation linéaire. Plus particulièrement, nous présentons des méthodes de plans coupants afin d'augmenter le taux de directions entières trouvées par ISUD, pour le rendre applicable aux instances industrielles de grande taille de type planification de personnel.

Généralisation de la formulation du problème d'augmentation et choix de la contrainte de normalisation. Au chapitre 4, nous reformulons le problème d'augmentation Mima, et en particulier les contraintes caractérisant son domaine de définition. Les contraintes linéaires, ainsi que les contraintes de bornes sont difficilement modifiables, car elles assurent la réalisabilité de la direction. Il en est de même pour l'objectif linéaire qui garantit que la direction est améliorante si et seulement si la solution optimale de C-Mima est strictement négative. Pour ces raisons, nous généralisons l'expression de la contrainte de normalisation, qui peut être modifiée tant que l'on s'assure qu'elle définit bien une section du cône des directions réalisables. Cette contrainte influence directement l'ensemble des solutions optimales ainsi que leur coût, tout en ne changeant pas le signe du coût de la direction : les directions de coût réduit négatif, quelle que soit la contrainte de normalisation, sont améliorantes (et réciproquement). Nous remplaçons donc la contrainte de normalisation $\boldsymbol{e}^T\boldsymbol{d}_{\mathcal{I}} = 1$ par sa version générique $\boldsymbol{w}^T\boldsymbol{d} = 1$ où le choix de $\boldsymbol{w}$ est laissé à l'utilisateur. Nous montrons que la direction réalisable déterminée par l'algorithme dépend fortement du choix des coefficients de cette contrainte ; il en va de même pour la probabilité que la solution vers laquelle elle mène soit entière. Nous étendons les propriétés théoriques liées à la décomposition dans l'algorihtme ISUD et montrons de nouveaux résultats dans le cas d'un choix de coefficients quelconques. Nous déterminons de nouvelles propriétés spécifiques à certains choix de normalisation et faisons des recommandations pour choisir les coefficients afin de pénaliser les directions fractionnaires au profit des directions entières. Des résultats numériques sur des instances de planification de personnel soulignent le potentiel de notre approche.

Amélioration de la relaxation linéaire C-Mima grâce à l'ajout de plans coupants. Au chapitre 5, nous nous penchons sur l'amélioration de la relaxation linéaire du problème d'augmentation. Une technique traditionnelle d'amélioration des relaxations linéaires en programmation en nombre entiers est l'utilisation de plans coupants qui permettent de séparer une solution optimale fractionnaire de la relaxation du domaine réalisable du problème en nombres entiers. Le problème d'augmentation entière iC-Mima n'est pas à proprement parler un programme linéaire en nombres entiers, mais son domaine est – comme pour PLNE – un ensemble discret. Savoir développer un algorithme de séparation pour iC-Mima permettrait donc d'améliorer la relaxation linéaire et potentiellement d'obtenir des solutions entières. Pour ce faire, nous caractérisons l'ensemble des inégalités valides pour iC-Mima comme l'ensemble non vide des coupes primales saturées par la solution courante $\boldsymbol{x}^0$, et proposons des procédures de séparation efficaces pour les coupes de cycles impairs et de cliques primales. Des résultats numériques prouvent la validité de notre approche ; ces tests sont effectués sur des instances de planification de personnel navigant et de chauffeurs d'autobus allant jusqu'à 1 600 contraintes et 570 000 variables. Sur les instances de transport aérien testées, l'ajout de coupes primales permet de passer d'un taux de résolution de 70% à 92%. Sur de grandes instances d'horaires de chauffeurs d'autobus, les coupes prouvent l'optimalité locale de la solution dans plus de 80% des cas.

Modification dynamique des poids de normalisation pour pénaliser les directions fractionnaires. Au chapitre 6, nous proposons un algorithme de modification dynamique des coefficients de la contrainte de normalisation de Mima. Nous proposons plusieurs stratégies de mise-à-jour de ces coefficients basées sur des observations théoriques et pratiques. La première de ces stratégies reprend directement les travaux exposés au chapitre 4, la seconde vise à pénaliser la direction fractionnaire déterminée par l'algorithme, la troisième tire parti des résultats du chapitre 5 en modifiant les coefficients de la contrainte de normalisation grâce à ceux de coupes primales. Cette version de l'algorithme est testée sur un nouvel ensemble d'instances provenant de l'industrie du transport aérien qui est, à notre connaissance, comparable à aucun autre. Il s'agit en effet de grands problèmes d'horaires de personnel navigant allant jusqu'à 1 700 vols et 115 000 rotations, donc autant de contraintes et de variables. Ils sont disponibles sous forme de problèmes de partitionnement pour lesquels nous fournissons des solutions initiales semblables à celles dont on disposerait en milieu industriel. Les résultats numériques obtenus montrent que l'algorithme est compétitif avec les méthodes conventionnelles basées sur le principe de séparation-et-évaluation.

Enfin, au chapitre 7 se trouve une discussion générale sur les apports et les limites de ce travail, et au chapitre 8 une conclusion.

CHAPITRE 4 ARTICLE 1: INFLUENCE OF THE NORMALIZATION CONSTRAINT ON THE INTEGRAL SIMPLEX USING DECOMPOSITION

Le texte de ce chapitre est celui de l'article *Influence of the Normalization Constraint on the Integral Simplex Using Decomposition* publié dans *Discrete Applied Mathematics* (Rosat et al., 2015a). Auteurs : Samuel Rosat, Driss Chakour, Issmail Elhallaoui, François Soumis.

Abstract

Since its introduction in 1969, the set partitioning problem has received much attention, and the structure of its feasible domain has been studied in detail. In particular, there exists a decreasing sequence of integer feasible points that leads to the optimum, such that each solution is a vertex of the polytope of the linear relaxation and adjacent to the previous one. Several algorithms are based on this observation and aim to determine that sequence; one example is the integral simplex using decomposition (ISUD) of Zaghrouti et al. (2014). In ISUD, the next solution is often obtained by solving a linear program without using any branching strategy. We study the influence of the normalization-weight vector of this linear program on the integrality of the next solution. We extend and strengthen the decomposition theory in ISUD, prove theoretical properties of the generic and specific normalization constraints, and propose new normalization constraints that encourage integral solutions. Numerical tests on scheduling instances (with up to 500,000 variables) demonstrate the potential of our approach.

Keywords 0–1 Programming, Set Partitioning Problem, Primal Algorithms, Normalization Constraint, Augmenting Algorithms

4.1 Introduction

Consider the set partitioning problem (SPP)

$$z^\star_{\text{SPP}} = \min_{\boldsymbol{x}} \left\{ \boldsymbol{c}^T \boldsymbol{x} \,|\, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{e}\,,\ \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{e}\,,\ \boldsymbol{x} \text{ is integer} \right\} \tag{SPP}$$

where $\boldsymbol{A} \in \{0,1\}^{m \times n}$ is an $m \times n$ binary matrix, and $\boldsymbol{c} \in \mathbb{N}^n$ is the cost vector. The vector of all zeros (resp. ones) with dimension dictated by the context is denoted $\boldsymbol{0}$ (resp. $\boldsymbol{e}$). Without loss of generality, we assume that $\boldsymbol{A}$ is full rank, contains no zero rows or columns, and has no identical rows or columns. $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{e}$ are called the linear constraints and $\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{e}$ the bound constraints. $\mathcal{F}_{\text{SPP}}$ denotes the set of all feasible solutions of SPP; $z^\star_{\text{SPP}}$ is called

the optimal value (or optimum) of SPP; and any feasible solution $\boldsymbol{x}^\star \in \mathcal{F}_{\text{SPP}}$ such that $\boldsymbol{c}^T\boldsymbol{x}^\star = z^\star_{\text{SPP}}$ is called an optimal solution of SPP. Note that, given the bounds ($\mathbf{0}$ and $\boldsymbol{e}$) and the integrality constraints, $\mathcal{F}_{\text{SPP}}$ contains only 0–1 vectors, i.e., $\mathcal{F}_{\text{SPP}} \subseteq \{0,1\}^n$. Finally, the linear relaxation of SPP, denoted SPP^{RL}, is the linear program obtained by removing the integrality constraints of SPP. Its feasible domain and optimal value are respectively denoted $\mathcal{F}_{\text{SPP}^{\text{RL}}}$ and $z^\star_{\text{SPP}^{\text{RL}}}$. Note that all optimization programs will be written in their minimization form, but the ideas and results also apply to maximization scenarios.

4.1.1 Integral Simplex Methods for the Set Partitioning Problem

Since its introduction by Garfinkel & Nemhauser (1969), the set partitioning problem has received much attention because of its wide range of applications: vehicle and crew scheduling, clustering problems, etc. It often appears within column-generation frameworks for such problems.

Many algorithms have been developed to solve SPP. As is the case for generic integer linear programs, they can be classified into three main classes (Letchford & Lodi, 2002): *dual fractional*, *dual integral*, and *primal* (or *augmentation*) methods. *Dual fractional* algorithms maintain optimality and linear-constraint feasibility at every iteration, and they stop when integrality is achieved. They are typically standard cutting plane procedures such as the algorithm of Gomory (1958). The classical branch-and-bound scheme is also based on a dual-fractional approach, in particular for the determination of lower bounds. *Dual integral* methods maintain integrality and optimality, and they terminate once the primal linear constraints are satisfied. Letchford & Lodi (2002) give a single example: another algorithm of Gomory (1963). Finally, *primal algorithms* maintain feasibility (and integrality) throughout the process and stop when optimality is reached. These are in fact descent algorithms for which the improving sequence $(\boldsymbol{x}_k)_{k=1\ldots K}$ satisfies the following conditions:

- `C1` $\boldsymbol{x}^k \in \mathcal{F}_{\text{SPP}}$;
- `C2` $\boldsymbol{x}^K$ is optimal;
- `C3` $\boldsymbol{c}^T\boldsymbol{x}^{k+1} < \boldsymbol{c}^T\boldsymbol{x}^k$.

A sequence satisfying the three conditions is called a sequence of *augmenting solutions* even in a minimization problem.

Given a current solution $\boldsymbol{x}^k \in \mathcal{F}_{\text{SPP}}$, primal algorithms are in fact based on the iterative

solution of the *augmentation problem* defined as:

AUG	Find an augmenting vector $\boldsymbol{y} \in \mathbb{N}^n$ s.t. $\boldsymbol{x}^{k+1} = (\boldsymbol{x}^k + \boldsymbol{y}) \in \mathcal{F}_{\text{SPP}}$ and $\boldsymbol{c}^T\boldsymbol{y} < 0$ or assert that $\boldsymbol{x}^k$ is optimal for SPP.

Traditionally, papers on constraint aggregation and integral simplex algorithms deal with minimization problems, whereas most authors present generic primal algorithms for maximization problems. We therefore draw the reader's attention to the following: **to retain the usual classification, we call the improvement problem AUG, although it supplies a decreasing direction**. For further information on primal algorithms in a general context, see the review of Spille & Weismantel (2005).

Two special features of SPP make it a particularly promising candidate for specialized primal methods. First, by definition, SPP is a 0–1 program, so every (integer) point of $\mathcal{F}_{\text{SPP}}$ is an extreme point (or vertex) of $\mathcal{F}_{\text{SPP}^{\text{RL}}}$. Thus, there exists a decreasing sequence of basic solutions satisfying conditions `C1`–`C3` . Second, as shown by Trubin (1969), SPP is *quasi-integral*, i.e., every edge of $Conv(\mathcal{F}_{\text{SPP}})$ is an edge of $\mathcal{F}_{\text{SPP}^{\text{RL}}}$. Thus, there exists a sequence of augmenting solutions such that the following condition holds:

`C4` $\boldsymbol{x}^{k+1}$ is an adjacent vertex of $\boldsymbol{x}^k$ in $\mathcal{F}_{\text{SPP}^{\text{RL}}}$.

An augmentation that satisfies `C4` is called an *integral augmentation*. Any sequence satisfying `C1`–`C4` is a sequence of *augmenting adjacent solutions*, and an algorithm that yields such a sequence is an *integral simplex*. Every solution can be obtained from the previous one in the sequence by performing one or several simplex pivots in SPP^{RL}, hence the name. Such sequences were first introduced by Balas & Padberg (1975). Several other authors (Haus et al., 2003; Thompson, 2002; Saxena, 2003a) have presented enumeration schemes that move from one integer solution to an adjacent one. An important recent contribution, in terms of both theory and applications, has been made by Rönnberg and Larsson (Rönnberg, 2012; Rönnberg & Larsson, 2009, 2014). However, none of these enumeration methods can guarantee a strict improvement (`C3`). They may perform degenerate pivots, in which there is no effective change in the solution (or the objective value) because the entering variable(s) takes the value zero. SPP tends to suffer severe degeneracy, so the computational time of these algorithms grows exponentially with the size of the instances.

Another pivot-based method proposed in the same spirit is the integral simplex using decomposition (ISUD) of Zaghrouti et al. (2014). It is one of the most promising recent developments, because it is based on linear programming techniques that take advantage of degeneracy and guarantee strict improvement at each iteration (Elhallaoui et al., 2011;

Omer et al., 2015). It follows an augmenting sequence of integer points leading to an optimal solution, such that each visited point is a vertex of $\mathcal{F}_{\text{SPPRL}}$ adjacent to the previous one (integral simplex). To find the edge leading to the next point, one solves a linear program to select an augmenting direction for the current point from a cone of feasible directions. To ensure that this linear program is bounded (the directions could go to infinity), a normalization constraint is added and the optimization is performed on a section of the cone. The solution of the linear program, i.e., the direction proposed by the algorithm, depends strongly on the chosen normalization weights, and so does the likelihood that the next solution is integer. In their seminal paper, Zaghrouti et al. (2014) consider all the weights to be equal to 1 in the normalization constraint; they therefore call it the *convexity* constraint. We extend the algorithm to the case of a generic normalization constraint, explore the theoretical properties of some specific constraints, and discuss the design of the normalization constraint based on our theoretical observations. We also report preliminary computational results that compare different normalization strategies and highlight their influence on the behavior of the algorithm.

4.1.2 Organization of the paper and contributions

This paper is organized as follows. In Section 4.2, we present a generalized version of the ISUD algorithm in the case of a generic normalization constraint. Moreover, we present the algorithm in an innovative primal form in which no dual considerations are taken into account, and we strengthen the theoretical results of Zaghrouti et al. (2014). We give new insight into the theoretical structure of the normalization and the structure of the sets of directions that can be followed to reach the next solution. We also show that, given any starting solution, it is always possible to obtain a sequence of solutions satisfying `C1`–`C4` without changing the normalization weights during the algorithm. Our primal point of view leads to geometric interpretations of most of the ideas. In Section 4.3, we propose four normalization strategies. Two are variants of classical strategies for which we prove theoretical properties, and the other two are based on our theoretical observations. In Section 4.4, we report numerical results for the four strategies that demonstrate the improvements obtained with our new normalization strategies in our version of ISUD. Section 4.5 provides concluding remarks. This paper provides a proof of concept based on innovative theoretical results; a follow-up paper will focus on the numerical aspects.

4.2 Integral Simplex Using Decomposition (ISUD): Generic Framework

As mentioned in the Introduction, several augmentation algorithms have been proposed for the SPP. Some of them satisfy only conditions `C1`, `C2`, and `C3`, while others also guarantee `C4`. It is possible to find integral augmentations that satisfy `C4` because the SPP is quasi-integral: there is an augmenting sequence of integral solutions $(x_k)_k$ such that for all k, $\boldsymbol{x}^{k+1}$ is a neighbor of $\boldsymbol{x}^k$ in $\mathcal{F}_{\mathrm{SPP}^{\mathrm{RL}}}$ and can be obtained by performing a sequence of simplex pivots.

4.2.1 Maximum Normalized Augmentation: Specifics and Linear Formulation

We first introduce some notation. For any polyhedron P, $Ext\,(P)$ is the set of its extreme points. We use lower-case bold symbols for column vectors and upper-case bold symbols for matrices. For the subsets $\mathcal{I} \subseteq \{1, \dots m\}$ of the row indices and $\mathcal{J} \subseteq \{1, \dots n\}$ of the column indices, the submatrix of $\boldsymbol{A}$ with rows indexed by $\mathcal{I}$ and columns indexed by $\mathcal{J}$ is denoted $\boldsymbol{A}_{\mathcal{I}\mathcal{J}}$. Similarly, $\boldsymbol{A}_{\mathcal{I}\boldsymbol{.}}$ is the set of rows of $\boldsymbol{A}$ indexed by $\mathcal{I}$, $\boldsymbol{A}_{\boldsymbol{.}\mathcal{J}}$ is the set of columns of $\boldsymbol{A}$ indexed by $\mathcal{J}$, and for any vector $\boldsymbol{v} \in \mathbb{R}^n$, $\boldsymbol{v}_{\mathcal{J}}$ is the subvector of all v_j, $j \in \mathcal{J}$. Finally, given any matrix $\boldsymbol{M}$, $\boldsymbol{M}^T$ is its transpose, and $Span(\boldsymbol{M})$ is the linear span of its columns, also called the image of $\boldsymbol{M}$. Hereinafter, assuming that k augmentations have been performed, $\boldsymbol{x}^k$ designates a known integer solution that we seek to improve, and $\boldsymbol{x}^{k+1}$ is the next solution that we want to determine. The sets of indices of the positive and zero-valued variables are respectively denoted as $\mathcal{P}^k$ and $\mathcal{Z}^k$ ($\mathcal{P}$ and $\mathcal{Z}$ when the context is clear). Hence, $\boldsymbol{x}^k_{\mathcal{P}} = \boldsymbol{e}$ and $\boldsymbol{x}^k_{\mathcal{Z}} = \boldsymbol{0}$. The set $\{1, \dots, n\}$ is thus partitioned into $(\mathcal{P}, \mathcal{Z})$. Set $\mathcal{P}$ is called the *reduced basis* and has cardinality $p = |\mathcal{P}|$. Without loss of generality, the columns of $\boldsymbol{A}$ are assumed to be ordered such that $\mathcal{P} = \{1, \dots, p\}$. In this paper, the terms basis, basic, and nonbasic refer to this reduced basis $\mathcal{P}$.

Remark. The main differences between $\mathcal{P}$ and a classical simplex basis are that p may be smaller than m, and an extreme solution of $\mathrm{SPP}^{\mathrm{RL}}$ is asociated with a single reduced basis instead of potentially many simplex bases. When $p < m$, the set of columns of $\boldsymbol{A}_{\boldsymbol{.}\mathcal{P}}$ is not a basis of $\mathbb{R}^m$ in the algebraic sense, but a linearly independent subset of $\mathbb{R}^m$ that spans only a subspace of $\mathbb{R}^m$, $Span(\boldsymbol{A}_{\boldsymbol{.}\mathcal{P}})$. In the classical simplex, the reduced basis is completed with columns from $\boldsymbol{A}$, forming a basis of $\mathbb{R}^m$. Entering a column of $\boldsymbol{A}$ of negative reduced cost which is within $Span(\boldsymbol{A}_{\boldsymbol{.}\mathcal{P}})$ into the basis produces a nondegenerate pivot that improves the solution. Such pivots are performed by working only on the reduced basis. However, they are not sufficient to reach optimality. The columns of $\boldsymbol{A}$ that are not in $Span(\boldsymbol{A}_{\boldsymbol{.}\mathcal{P}})$ must also be considered. Another way to improve the solution is to find an improving direction in $Span(\boldsymbol{A}_{\boldsymbol{.}\mathcal{P}})$ combining such variables so as to perform nondegenerates pivots. If the reduced

cost of the combination is negative, the solution is hence improved. More about degeneracy and reduced bases can be found in the work of Omer et al. (2015).

To formulate **AUG** in a solvable way, we define $\boldsymbol{y} = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k$ and introduce the following definition:

Definition 1. A direction $\boldsymbol{d} \in \mathbb{R}^n$ is *feasible* at $\boldsymbol{x}^k$ if there exists a step $\rho > 0$ such that $\boldsymbol{x}^k + \rho\boldsymbol{d} \in \mathcal{F}_{\mathrm{SPP}^{\mathrm{RL}}}$, and it is *augmenting* if $\boldsymbol{c}^T\boldsymbol{d} < 0$. Moreover, given a feasible direction, we denote the largest feasible step by $r(\boldsymbol{d}) = \max\left\{\rho \in \mathbb{R} | \boldsymbol{x}^k + \rho\boldsymbol{d} \in \mathcal{F}_{\mathrm{SPP}^{\mathrm{RL}}}\right\} \geq 0$.

Ignoring the integrality constraints of SPP, we can therefore characterize an augmenting vector $\boldsymbol{y}$ in $\mathrm{SPP}^{\mathrm{RL}}$ by a pair $(\boldsymbol{d}, \rho) \in \mathbb{R}^n \times \mathbb{R}$, where $\boldsymbol{d}$ can be normalized at will, as long as $\boldsymbol{x}^k + \boldsymbol{y} = \boldsymbol{x}^k + \rho\boldsymbol{d} \in \mathcal{F}_{\mathrm{SPP}^{\mathrm{RL}}}$. The set of all feasible directions can be described as the following cone:

$$\Gamma = \{\boldsymbol{d} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{d} = 0\,,\, \boldsymbol{d}_{\mathcal{P}} \leq \boldsymbol{0}\,,\, \boldsymbol{d}_{\mathcal{Z}} \geq \boldsymbol{0}\} \tag{4.1}$$

from which we will only consider a section,

$$\Delta_{\boldsymbol{w}} = \left\{\boldsymbol{d} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{d} = 0\,,\, \boldsymbol{w}^T\boldsymbol{d} = 1\,,\, \boldsymbol{d}_{\mathcal{P}} \leq \boldsymbol{0}\,,\, \boldsymbol{d}_{\mathcal{Z}} \geq \boldsymbol{0}\right\} \tag{4.2}$$

where $\boldsymbol{w}$ is the normalization vector. A geometric interpretation of Γ and $\Delta_{\boldsymbol{w}}$ is given in Figure 4.1.

In this paper, we will always assume that $\boldsymbol{w}_{\mathcal{P}} \leq \boldsymbol{0}$ and $\boldsymbol{w}_{\mathcal{Z}} > \boldsymbol{0}$, which are sufficient conditions for the cone section to be well defined. Indeed, in the general case, this ensures that whenever $\boldsymbol{d}$ is feasible, $\boldsymbol{w}^T\boldsymbol{d} > 0$ (because we have $\boldsymbol{A}\boldsymbol{d} = 0$, $\boldsymbol{d}_{\mathcal{Z}} = 0$ implies $\boldsymbol{d}_{\mathcal{P}} = 0$). Moreover, setting any of the $w_j = 0$ for $j \in \mathcal{Z}$ could make the problem unbounded. These conditions therefore seem sufficient without being restrictive. The (fractional) augmentation problem can then be formulated as the following linear program, called the *maximum normalized augmentation*:

$$z^{\star}_{\mathrm{MNA}^{\boldsymbol{w}}} = \min\left\{\boldsymbol{c}^T\boldsymbol{d} \,|\, \boldsymbol{d} \in \Delta_{\boldsymbol{w}}\right\} \tag{MNA$^{\boldsymbol{w}}$}$$

and the following proposition holds.

Proposition 1. *$\boldsymbol{x}^k$ is optimal for $\mathrm{SPP}^{\mathrm{RL}}$ if and only if $z^{\star}_{\mathrm{MNA}^{\boldsymbol{w}}} \geq 0$.*

Proof. This proposition is a specialization of Proposition 1 of Omer et al. (2015). The proof is based on the following argument: on the one hand, if there exists an improving direction, taking a small step in this direction leads to a strictly better solution and the current one is nonoptimal. On the other hand, if the solution is nonoptimal, then any vector that leads

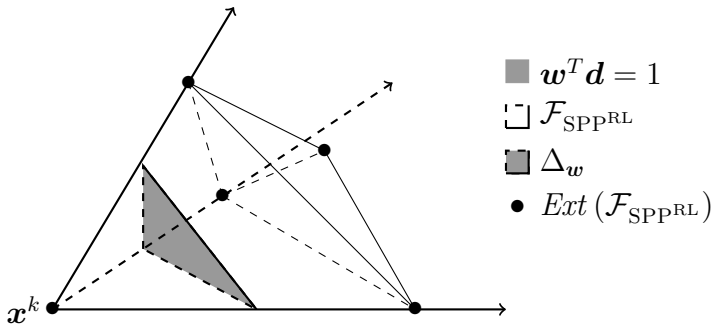


Figure 4.1 Geometric description of $\Delta_{\boldsymbol{w}}$. The cone of all feasible directions at $\boldsymbol{x}^k$ is defined by the three arrowed lines. The grey area represents $\Delta_{\boldsymbol{w}}$, i.e., the set of normalized feasible directions.

to a strictly better solution can be normalized so that it becomes normalized, augmenting, and feasible. □

The effect of a normalization constraint is to maximize the marginal cost of a direction according to the weight vector $\boldsymbol{w}$. The geometric interpretation is shown in Figure 4.2: $\boldsymbol{d}^{1,1}$ and $\boldsymbol{d}^{2,2}$ are respectively optimal for the normalization weights $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. However, they respectively lead to $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$, which are different adjacent vertices of $\boldsymbol{x}^k$ in $\mathcal{F}_{\text{SPP}^{\text{RL}}}$. Observe that in this example, $\boldsymbol{x}^1$ is integer, but $\boldsymbol{x}^2$ is fractional, showing that the normalization can influence the integrality of the next solution. Note also that $\boldsymbol{x}^4$ does not correspond to any extreme point of $\Delta_{\boldsymbol{w}}$, because it is not adjacent to $\boldsymbol{x}^k$.

Finally,

$$\text{MNA}^{\boldsymbol{w}} \Leftrightarrow \min\left\{\frac{\boldsymbol{c}^T\boldsymbol{d}}{\boldsymbol{w}^T\boldsymbol{d}} \,|\, \boldsymbol{d} \in \Gamma\right\}. \tag{4.3}$$

In this equivalent nonlinear formulation, the domain is unbounded (cone Γ). However, the cost is constant for each direction of the cone, which ensures that formulation (4.3) is bounded.

4.2.2 Row-reduction of MNA$^{\boldsymbol{w}}$

In this section, we will show that p constraints and p variables can be eliminated from the linear program MNA$^{\boldsymbol{w}}$ if a suitable transformation on the constraint matrix is performed. This technique derives from the constraint aggregation methods first introduced by Elhallaoui et al. (Elhallaoui et al. (2005) for SPP and Elhallaoui et al. (2011) in a more general context). Only the nonbasic variables of $\mathcal{Z}$ are kept in the problem, the costs become the

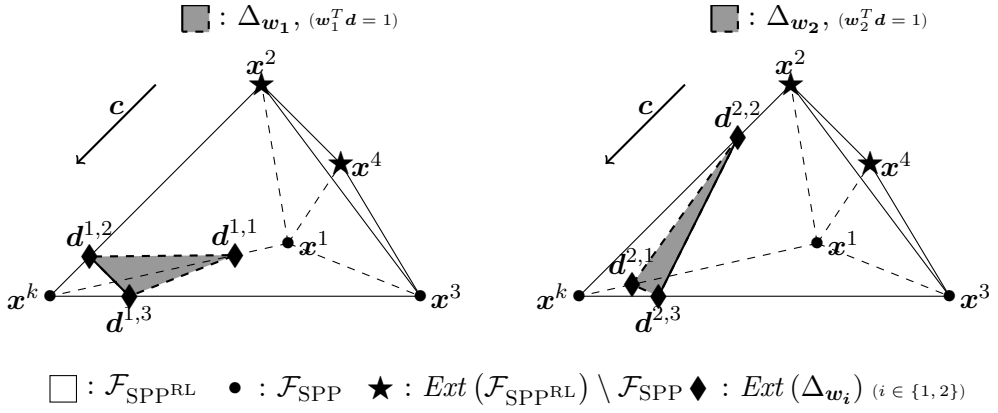


Figure 4.2 Example of two different normalization constraints for the same problem. The polyhedra represent $\mathcal{F}_{\mathrm{SPP}^{\mathrm{RL}}}$, the feasible domain of $\mathrm{SPP}^{\mathrm{RL}}$. The dots (•) are the integer extreme points (i.e., $\mathcal{F}_{\mathrm{SPP}}$), and the stars (★) are the fractional extreme points. The grey polyhedra are the cone sections $\Delta_{\boldsymbol{w_i}}$, $i \in \{1,2\}$, and the diamonds are their extreme points.

reduced costs computed as in the simplex algorithm, and the constraints are modified accordingly. Since $\boldsymbol{A}$ contains no zero columns, for each column $j \in \mathcal{P}$ (the indices of the nonzero variables of the current solution), there exists $i_j \in \{1,\ldots,m\}$ such that $A_{i_j j} = 1$ (i_j may not be unique). Without loss of generality, given such an i_j the rows of $\boldsymbol{A}$ can be reordered so that $i_j = j$ for all $j \in \mathcal{P} = \{1,\ldots,p\}$. Moreover, recall that $\boldsymbol{A}$ is a $\{0,1\}$ matrix, SPP has only equality constraints, and the right-hand side of each of the equality constraints is equal to 1. Therefore, the set of variables taking the value 1 in the current integer solution $\boldsymbol{x}^k$ forms a partition of the rows in the following sense: for each row i, there exists a unique column $j \in \mathcal{P}$ such that $A_{ij} = 1$. Thus, for any $j \in \mathcal{P}$, $A_{i_j j}$ is the only nonzero entry in row i in $\boldsymbol{A}_{\cdot\mathcal{P}}$. Hence, the constraint matrix can be partitioned as follows:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I}_p & \boldsymbol{A}_{\mathcal{P}\mathcal{Z}} \\ \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}} & \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{Z}} \end{bmatrix} \tag{4.4}$$

where $\bar{P} = \{1,\ldots m\} \setminus \mathcal{P}$ and $\boldsymbol{I}_p$ is the $p \times p$ identity matrix. With this partition, we can form a projection $\bar{\Delta}_{\boldsymbol{w}}$ of $\Delta_{\boldsymbol{w}}$ on $\mathbb{R}^{n-p}$ using the projection matrix $\boldsymbol{T} = \begin{bmatrix} -\boldsymbol{A}_{\mathcal{P}\mathcal{Z}} \\ \boldsymbol{I}_{n-p} \end{bmatrix}$, called the *transformation* matrix. We define the reduced-weight vector to be $\bar{\boldsymbol{w}}^T = \boldsymbol{w}^T\boldsymbol{T} = -\boldsymbol{w}_{\mathcal{P}}^T\boldsymbol{A}_{\mathcal{P}\mathcal{Z}} + \boldsymbol{w}_{\mathcal{Z}}^T$ and the reduced matrix to be $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}} = \boldsymbol{A}_{\bar{\mathcal{P}}\cdot}\boldsymbol{T} = -\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{Z}} + \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{Z}}$. Then, $\bar{\Delta}_{\boldsymbol{w}} = \left\{\boldsymbol{d}_{\mathcal{Z}} \in \mathbb{R}^{n-p} \mid \bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}}\boldsymbol{d}_{\mathcal{Z}} = 0\,,\ \bar{\boldsymbol{w}}^T\boldsymbol{d}_{\mathcal{Z}} = 1\,,\ \boldsymbol{d}_{\mathcal{Z}} \geq \boldsymbol{0}\right\}$. Proposition 2 below states explicitly the linear relationship between $\Delta_{\boldsymbol{w}}$ and $\bar{\Delta}_{\boldsymbol{w}}$: it shows that $\Delta_{\boldsymbol{w}}$ is a linear lifting of $\bar{\Delta}_{\boldsymbol{w}}$ in $\mathbb{R}^n$.

Remark. The matrix $\bar{\boldsymbol{A}}$ is, in a sense, the equivalent of the simplex tableau matrix in the case of a reduced basis. Indeed, if the working basis $\boldsymbol{A}_{\cdot\mathcal{P}}$ is completed as a full basis $\boldsymbol{B} =$

$\begin{bmatrix} \boldsymbol{I}_p & \boldsymbol{0} \\ \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}} & \boldsymbol{I}_{m-p} \end{bmatrix}$, then $\bar{\boldsymbol{A}} = \boldsymbol{B}^{-1}\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I}_p & \boldsymbol{0} \\ \boldsymbol{0} & \bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}} \end{bmatrix}$. Unlike in the simplex algorithm, the columns that are chosen to complete $\boldsymbol{A}_{\cdot\mathcal{P}}$ as a basis $\boldsymbol{B}$ do not necessarily originate from $\boldsymbol{A}_{\cdot\mathcal{Z}}$, and there is an infinite number of possible completions. Obviously, the choice of the matrix $\boldsymbol{B}$, called the *compatibility matrix*, strongly influences the value of $\bar{\boldsymbol{A}}$ and thus the behavior of the algorithm. See Bouarab et al. (2014) for more details on the role played by the compatibility matrix. In this paper, we consider only the case where $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}} = \boldsymbol{A}_{\bar{\mathcal{P}}\cdot}\boldsymbol{T} = -\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{Z}} + \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{Z}}$.

Proposition 2. $\Delta_{\boldsymbol{w}} = \left\{\boldsymbol{d} = \boldsymbol{T}\boldsymbol{d}_{\mathcal{Z}} \,|\, \boldsymbol{d}_{\mathcal{Z}} \in \bar{\Delta}_{\boldsymbol{w}}\right\}$.

Proof. Let $\boldsymbol{d} \in \mathbb{R}^n$. Given Equation (4.4) and since $\boldsymbol{A}$ is a binary matrix,

$$
\begin{aligned}
\boldsymbol{d} \in \Delta_{\boldsymbol{w}} &\Leftrightarrow \boldsymbol{A}\boldsymbol{d} = 0, \boldsymbol{w}^T\boldsymbol{d} = 1, \boldsymbol{d}_{\mathcal{P}} \leq \boldsymbol{0}, \boldsymbol{d}_{\mathcal{Z}} \geq 0 \\
&\Leftrightarrow \begin{cases}
\boldsymbol{d}_{\mathcal{P}} & +\boldsymbol{A}_{\mathcal{P}\mathcal{Z}}\boldsymbol{d}_{\mathcal{Z}} & = \boldsymbol{0} \\
\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{d}_{\mathcal{P}} & +\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{Z}}\boldsymbol{d}_{\mathcal{Z}} & = \boldsymbol{0} \\
\boldsymbol{w}_{\mathcal{P}}^T\boldsymbol{d}_{\mathcal{P}} & +\boldsymbol{w}_{\mathcal{Z}}^T\boldsymbol{d}_{\mathcal{Z}} & = 1 \\
\boldsymbol{d}_{\mathcal{P}} \leq 0 & \boldsymbol{d}_{\mathcal{Z}} \geq 0 &
\end{cases} \\
&\Leftrightarrow \begin{cases}
\boldsymbol{d}_{\mathcal{P}} & = -\boldsymbol{A}_{\mathcal{P}\mathcal{Z}}\boldsymbol{d}_{\mathcal{Z}} \\
\left(-\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{Z}} + \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{Z}}\right)\boldsymbol{d}_{\mathcal{Z}} & = \boldsymbol{0} \\
\left(-\boldsymbol{w}_{\mathcal{P}}^T\boldsymbol{A}_{\mathcal{P}\mathcal{Z}} + \boldsymbol{w}_{\mathcal{Z}}^T\right)\boldsymbol{d}_{\mathcal{Z}} & = 1 \\
\boldsymbol{d}_{\mathcal{Z}} \geq 0 &
\end{cases} \\
&\Leftrightarrow \boldsymbol{d} = \boldsymbol{T}\boldsymbol{d}_{\mathcal{Z}} \text{ and } \boldsymbol{d}_{\mathcal{Z}} \in \bar{\Delta}_{\boldsymbol{w}}.
\end{aligned}
$$

□

Proposition 2 shows that any feasible direction $\boldsymbol{d}$ can be partitioned as $\boldsymbol{d} = (\boldsymbol{d}_{\mathcal{P}}, \boldsymbol{d}_{\mathcal{Z}}) = (-\boldsymbol{A}_{\mathcal{P}\mathcal{Z}}\boldsymbol{d}_{\mathcal{Z}}, \boldsymbol{d}_{\mathcal{Z}}) = \boldsymbol{T}\boldsymbol{d}_{\mathcal{Z}}$. Regarding the cost of such a direction, $\boldsymbol{c}^T\boldsymbol{d} = \boldsymbol{c}^T\boldsymbol{T}\boldsymbol{d}_{\mathcal{Z}}$. Write $\bar{\boldsymbol{c}} = \boldsymbol{T}^T\boldsymbol{c}$, i.e., $\bar{\boldsymbol{c}}^T = -\boldsymbol{c}_{\mathcal{P}}^T\boldsymbol{A}_{\mathcal{P}\mathcal{Z}} + \boldsymbol{c}_{\mathcal{Z}}^T$, then MNA$^{\boldsymbol{w}}$ is equivalent to the following program, called the *reduced* MNA$^{\boldsymbol{w}}$:

$$
z^{\star}_{\text{R-MNA}^{\boldsymbol{w}}} = \min\left\{\bar{\boldsymbol{c}}^T\boldsymbol{d}_{\mathcal{Z}} \,|\, \boldsymbol{d}_{\mathcal{Z}} \in \bar{\Delta}_{\boldsymbol{w}}\right\} \qquad \text{(R-MNA}^{\boldsymbol{w}}\text{)}
$$

The costs of R-MNA$^{\boldsymbol{w}}$ are in fact the reduced costs associated with the variables that are not in the working basis: their own marginal cost ($\boldsymbol{c}_{\mathcal{Z}}^T$), minus the marginal impact they have on the values of the variables from $\mathcal{P}$ if they enter the solution ($-\boldsymbol{w}_{\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{Z}}$). Interestingly, although not surprisingly, the same goes for the normalization constraint that undergoes the same linear transformation ($\bar{\boldsymbol{w}} = \boldsymbol{T}^T\boldsymbol{w}$). Proposition 2 describes only a global connection between $\Delta_{\boldsymbol{w}}$ and $\bar{\Delta}_{\boldsymbol{w}}$; this can be strengthened to a one-to-one correspondence:

Proposition 3. *Given normalization weights $\boldsymbol{w}$ in $\Delta_{\boldsymbol{w}}$ and the corresponding "reduced*

weights" $\bar{\boldsymbol{w}} = \boldsymbol{T}^T\boldsymbol{w}$ *in* $\bar{\Delta}_{\boldsymbol{w}}$*, the function*

$$\begin{array}{rccl} T: & \bar{\Delta}_{\boldsymbol{w}} \subseteq \mathbb{R}^{n-p} & \rightarrow & \Delta_{\boldsymbol{w}} \subseteq \mathbb{R}^n \\ & \boldsymbol{d}_{\mathcal{Z}} & \mapsto & \boldsymbol{d} = \boldsymbol{T}\boldsymbol{d}_{\mathcal{Z}} = (-\boldsymbol{A}_{\mathcal{P}\mathcal{Z}}\boldsymbol{d}_{\mathcal{Z}}, \boldsymbol{d}_{\mathcal{Z}}) \end{array} \tag{4.5}$$

is a bijective function. It maps every vertex of $\bar{\Delta}_{\boldsymbol{w}}$ *to a vertex of* $\Delta_{\boldsymbol{w}}$ *with the same cost* ($\bar{\boldsymbol{c}}^T\boldsymbol{d}_{\mathcal{Z}} = \boldsymbol{c}^T\boldsymbol{d}$)*, and vice versa.*

Proof. This follows from the previous results and the fact that the image of a polyhedron after a linear transformation is a polyhedron whose vertices are the images of the vertices of the original polyhedron. □

Figure 4.3 gives a geometric interpretation of Proposition 3. As a consequence of the previous results, the following corollary holds.

Corollary 4. $\boldsymbol{x}^k$ *is optimal for* $\mathrm{SPP}^{\mathrm{RL}}$ *if and only if* $z^\star_{\text{R-MNA}^{\boldsymbol{w}}} \geq 0$*. Moreover,* $\boldsymbol{d}^\star_{\mathcal{Z}}$ *is an optimal solution of* R-MNA$^{\boldsymbol{w}}$ *if and only if* $\boldsymbol{d}^\star = \boldsymbol{T}\boldsymbol{d}^\star_{\mathcal{Z}}$ *is an optimal solution of* MNA$^{\boldsymbol{w}}$*.*

From Proposition 3, we infer that if there exists an improving feasible direction, it can be expressed as the transformation $\boldsymbol{T}\boldsymbol{d}_{\mathcal{Z}}$ of any feasible solution $\boldsymbol{d}_{\mathcal{Z}} \in \bar{\Delta}_{\boldsymbol{w}}$ of negative reduced cost. Moreover, Corollary 4 shows that solving R-MNA$^{\boldsymbol{w}}$ is sufficient to determine a feasible direction if one exists. The question of the fractional augmentation is thus equivalent to the question $z^\star_{\text{R-MNA}^{\boldsymbol{w}}} < 0$?

Finally, note that MNA$^{\boldsymbol{w}}$ and MNA$^{(\boldsymbol{0},\bar{\boldsymbol{w}})}$ are strictly equivalent as $\overline{(\boldsymbol{0},\bar{\boldsymbol{w}})} = \bar{\boldsymbol{w}}$ and therefore R-MNA$^{\boldsymbol{w}} \Leftrightarrow$ R-MNA$^{(\boldsymbol{0},\bar{\boldsymbol{w}})}$. This shows *a posteriori* that we could also have assumed $\boldsymbol{w}_{\mathcal{P}} = 0$ from the beginning, without any loss of generality. The normalization weights therefore need to apply only to the incoming variables and, when this is not the case, an equivalent weight vector $(\boldsymbol{0},\bar{\boldsymbol{w}})$ can be easily determined. That condition still guarantees $\boldsymbol{w}_{\mathcal{Z}} > 0$ since $\boldsymbol{w}_{\mathcal{I}} = \bar{\boldsymbol{w}} = -\boldsymbol{A}_{\mathcal{P}\mathcal{Z}}\boldsymbol{w}_{\mathcal{P}} + \boldsymbol{w}_{\mathcal{Z}}$ and all coefficients of $\boldsymbol{A}$ are nonnegative, and $\boldsymbol{w}_{\mathcal{P}} \leq \boldsymbol{0}$.

Corollary 5. $\Delta_{\boldsymbol{w}} = \Delta_{\boldsymbol{w}'}$ *where* $\boldsymbol{w}' = (\boldsymbol{0},\bar{\boldsymbol{w}})$ *with* $\boldsymbol{w}'_{\mathcal{P}} = \boldsymbol{0}$*.*

Hereinafter, unless stated otherwise, we will assume that $\boldsymbol{w}_{\mathcal{P}} = 0$, i.e, $\boldsymbol{w} = (\boldsymbol{0},\boldsymbol{w}_{\mathcal{Z}})$. In the same way that we proposed a nonlinear equivalent to the full augmentation problem, we can rewrite R-MNA$^{\boldsymbol{w}}$ as

$$\text{R-MNA}^{\boldsymbol{w}} \Leftrightarrow \min\left\{\frac{\bar{\boldsymbol{c}}^T\boldsymbol{d}_{\mathcal{Z}}}{\bar{\boldsymbol{w}}^T\boldsymbol{d}_{\mathcal{Z}}} \,|\, \boldsymbol{d} \in \bar{\Gamma}\right\}, \tag{4.6}$$

where $\bar{\Gamma} = \left\{\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}}\boldsymbol{d}_{\mathcal{Z}} = 0\,,\, \boldsymbol{d}_{\mathcal{Z}} \geq \boldsymbol{0}\right\}$ is the cone of all normalized feasible directions. The domain of this nonlinear formulation is again unbounded, but formulation (4.6) is bounded because the reduced cost is constant for each direction of the cone.

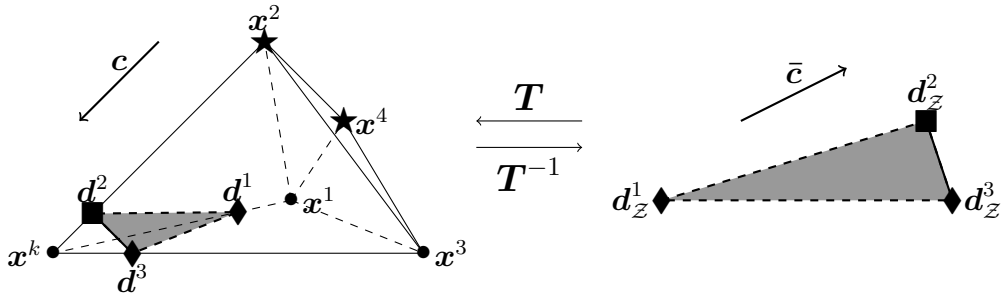


Figure 4.3 Geometric interpretation of transformation $\boldsymbol{T}$. The original space $\mathbb{R}^n$ with the feasible polyhedron $\mathcal{F}_{\text{SPP}^{\text{RL}}}$ is shown on the left. Bullets ($\bullet$) indicate the integer extreme points ($\mathcal{F}_{\text{SPP}}$) and stars ($\bigstar$) indicate the fractional extreme points. The set of normalized feasible directions $\Delta_{\boldsymbol{w}}$ is shown in grey, and its extreme points are either integral ($\blacklozenge$) or fractional ($\blacksquare$) directions. The image of $\Delta_{\boldsymbol{w}}$ under $\boldsymbol{T}^{-1}$ in $\mathbb{R}^{n-p}$ is shown on the right. Diamonds ($\blacklozenge$) indicate the integer reduced directions and squares ($\blacksquare$) indicate the fractional reduced directions. The costs ($\boldsymbol{c} \in \mathbb{R}^n$) and reduced costs ($\bar{\boldsymbol{c}}$) are indicated. The solution of R-MNA$^{\boldsymbol{w}}$ is $\boldsymbol{d}^1_{\mathcal{Z}}$, corresponding to the integral direction $\boldsymbol{d}^1 = \boldsymbol{T}\boldsymbol{d}^1_{\mathcal{Z}}$.

4.2.3 Geometric Structure of $\bar{\Delta}_{\boldsymbol{w}}$

The previous sections give a way to find a fractional augmentation. Given $\boldsymbol{d} \in \Delta_{\boldsymbol{w}}$, $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + r(\boldsymbol{d})\boldsymbol{d} \in \mathcal{F}_{\text{SPP}^{\text{RL}}}$ may indeed be fractional and not in $\mathcal{F}_{\text{SPP}}$. If it is integer, $\boldsymbol{d}$ (or $\boldsymbol{d}_{\mathcal{Z}}$) is called an *integral direction*; otherwise it is *fractional*. In Sections 4.2.3 and 4.2.4, we characterize these integer directions and show that $\boldsymbol{d}$ is integral provided $\boldsymbol{w}$ is chosen well.

A solution $\boldsymbol{d}_{\mathcal{Z}}$ of R-MNA$^{\boldsymbol{w}}$ is called *minimal* if there exists no other solution of R-MNA$^{\boldsymbol{w}}$ whose support is strictly contained in that of $\boldsymbol{d}$. Terms such as *nondecomposable* (Balas & Padberg, 1975) or *irreducible* (Haus et al., 2003) are also used to describe such directions. Lemma 6 below will be used in the next section and shows that, provided we use the simplex algorithm, the solution of R-MNA$^{\boldsymbol{w}}$ is indeed minimal.

Lemma 6. *Ext* $\left(\bar{\Delta}_{\boldsymbol{w}}\right)$ *is the set of minimal directions of* $\bar{\Delta}_{\boldsymbol{w}}$.

Proof. First, let $\boldsymbol{d}^\star_{\mathcal{Z}} \in \mathit{Ext}\left(\bar{\Delta}_{\boldsymbol{w}}\right)$; we will prove that it is minimal. Let $\boldsymbol{u}_{\mathcal{Z}} \in \bar{\Delta}_{\boldsymbol{w}}$ such that $\mathit{Supp}(\boldsymbol{u}_{\mathcal{Z}}) \subseteq \mathit{Supp}(\boldsymbol{d}^\star_{\mathcal{Z}})$. Moreover, both $\boldsymbol{u}_{\mathcal{Z}} \geq 0$ and $\boldsymbol{d}^\star_{\mathcal{Z}} \geq 0$, so there exists $\epsilon \in (0,1)$ such that $\boldsymbol{v}_{\mathcal{Z}} = \boldsymbol{d}^\star_{\mathcal{Z}} - \epsilon \boldsymbol{u}_{\mathcal{Z}} \geq 0$. Let $\boldsymbol{v}'_{\mathcal{Z}} = \boldsymbol{v}_{\mathcal{Z}} / (1-\epsilon)$. $\boldsymbol{v}'_{\mathcal{Z}}$ satisfies $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}}\boldsymbol{v}'_{\mathcal{Z}} = \frac{1}{1-\epsilon}\left(\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}}\boldsymbol{d}^\star_{\mathcal{Z}} - \epsilon \bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}}\boldsymbol{u}_{\mathcal{Z}}\right) = \boldsymbol{0}$, $\bar{\boldsymbol{w}}^T\boldsymbol{v}'_{\mathcal{Z}} = \frac{\bar{\boldsymbol{w}}^T\boldsymbol{d}^\star_{\mathcal{Z}} - \epsilon\bar{\boldsymbol{w}}^T\boldsymbol{u}_{\mathcal{Z}}}{1-\epsilon} = \frac{1-\epsilon}{1-\epsilon} = 1$, and $\boldsymbol{v}'_{\mathcal{Z}} \geq 0$. Hence, $\boldsymbol{v}'_{\mathcal{Z}} \in \bar{\Delta}_{\boldsymbol{w}}$ and $\boldsymbol{d}^\star_{\mathcal{Z}} = \epsilon\boldsymbol{u}_{\mathcal{Z}} + (1-\epsilon)\,\boldsymbol{v}'_{\mathcal{Z}}$ is a convex combination of two elements of $\bar{\Delta}_{\boldsymbol{w}}$. Since $\boldsymbol{d}^\star_{\mathcal{Z}} \in \mathit{Ext}\left(\bar{\Delta}_{\boldsymbol{w}}\right)$, we have $\boldsymbol{u}_{\mathcal{Z}} = \boldsymbol{v}'_{\mathcal{Z}} = \boldsymbol{d}_{\mathcal{Z}}$.

Now, let $\boldsymbol{d}^\star_{\mathcal{Z}} \in \bar{\Delta}_{\boldsymbol{w}}$ be a minimal direction; we will prove that $\boldsymbol{d}^\star_{\mathcal{Z}} \in \mathit{Ext}\left(\bar{\Delta}_{\boldsymbol{w}}\right)$. Let $\epsilon > 0$, $\boldsymbol{u}_{\mathcal{Z}}$,

and $\boldsymbol{v}_{\mathcal{Z}}$ be such that

$$\begin{cases} \boldsymbol{d}^{\star}_{\mathcal{Z}} = \epsilon \boldsymbol{u}_{\mathcal{Z}} + (1-\epsilon)\, \boldsymbol{v}_{\mathcal{Z}} \\ \boldsymbol{u}_{\mathcal{Z}}, \boldsymbol{v}_{\mathcal{Z}} \in \bar{\Delta}_{\boldsymbol{w}} \end{cases}$$

and suppose $\boldsymbol{u}_{\mathcal{Z}} \neq \boldsymbol{d}^{\star}_{\mathcal{Z}}$, $\boldsymbol{d}^{\star}_{\mathcal{Z}} \neq \boldsymbol{d}^{\star}_{\mathcal{Z}}$. Since $\boldsymbol{d}^{\star}_{\mathcal{Z}}$ is minimal, $Supp(\boldsymbol{u}_{\mathcal{Z}})$ and $Supp(\boldsymbol{v}_{\mathcal{Z}})$ cannot be contained in $Supp(\boldsymbol{d}^{\star}_{\mathcal{Z}})$, so there exists $j \notin Supp(\boldsymbol{d}^{\star}_{\mathcal{Z}})$ such that $\epsilon u_j + (1-\epsilon)\, v_j = 0$ and $u_j \geq 0$ and $v_j \neq 0$. Hence, either u_j or v_j is negative, which contradicts $\boldsymbol{u}_{\mathcal{Z}}, \boldsymbol{v}_{\mathcal{Z}} \in \bar{\Delta}_{\boldsymbol{w}}$. Therefore, $\boldsymbol{d}^{\star}_{\mathcal{Z}} = \boldsymbol{u}_{\mathcal{Z}} = \boldsymbol{v}_{\mathcal{Z}}$ and $\boldsymbol{d}^{\star}_{\mathcal{Z}} \in Ext\left(\bar{\Delta}_{\boldsymbol{w}}\right)$. □

4.2.4 Normalization and Integrality

Let us now consider the integrality of the directions. Let $\bar{\Delta}^{int}_{\boldsymbol{w}}$ (or $\Delta^{int}_{\boldsymbol{w}}$) be the set of integral directions at $\boldsymbol{x}^k$. The quasi-integral property of SPP yields

$$\bar{\Delta}^{int}_{\boldsymbol{w}} \subseteq Conv(\bar{\Delta}^{int}_{\boldsymbol{w}} \cap Ext\left(\bar{\Delta}_{\boldsymbol{w}}\right)), \tag{4.7}$$

i.e., the integral directions are contained in the convex hull of those that are also extreme points of $\bar{\Delta}_{\boldsymbol{w}}$. Therefore, we can restrict the search for an augmenting integral direction to the extreme points of $\bar{\Delta}_{\boldsymbol{w}}$. Proposition 7 gives a simple way to test whether or not $\boldsymbol{d}^{\star}_{\mathcal{Z}} \in Ext\left(\bar{\Delta}_{\boldsymbol{w}}\right)$ is also in $\bar{\Delta}^{int}_{\boldsymbol{w}}$.

Definition 2. Given a set of indices of the variables $\mathcal{S} \subseteq \{1, \dots n\}$, it is *column-disjoint* if no pair of columns of $\boldsymbol{A}_{\cdot\mathcal{S}}$ has a common nonzero entry, i.e., $\forall i, j \in \mathcal{S}, i \neq j, \forall k \in \{1, \dots m\}, A_{ki}A_{kj} = 0$. This definition is extended to $\boldsymbol{A}_{\cdot\mathcal{S}}$ and $\boldsymbol{x}_{\mathcal{S}}$. Since $\boldsymbol{A}$ is binary, $\mathcal{S}$ is column-disjoint iff $\boldsymbol{A}_{\cdot\mathcal{S}}$ is a set of orthogonal columns.

Proposition 7. *Given* $\boldsymbol{d}^{\star}_{\mathcal{Z}} \in Ext\left(\bar{\Delta}_{\boldsymbol{w}}\right)$ *and* $\mathcal{S} = Supp(\boldsymbol{d}^{\star}_{\mathcal{Z}})$,

$$\boldsymbol{d}^{\star}_{\mathcal{Z}} \text{ is an integral direction} \Leftrightarrow \mathcal{S} \text{ is column-disjoint.} \tag{4.8}$$

For such a direction, $d^{\star}_j = 1/\bar{W}_{\mathcal{S}}$ *if* $j \in \mathcal{S}$ *and* 0 *otherwise, where* $\bar{W}_{\mathcal{S}} = \sum_{j \in \mathcal{S}} \bar{w}_j$. *Also,* $r(\boldsymbol{d}_{\mathcal{Z}}) = \bar{W}_{\mathcal{S}}$.

Proof. Zaghrouti et al. (2014) proved this proposition in the particular case of $\bar{\boldsymbol{w}} = \boldsymbol{e}$. We will show that, whatever the normalization constraint, we can reduce the problem to this case. Let $\boldsymbol{d}^{\star}_{\mathcal{Z}} \in Ext\left(\bar{\Delta}_{\boldsymbol{w}}\right)$, $\mathcal{S} = Supp(\boldsymbol{d}^{\star}_{\mathcal{Z}})$, and $\boldsymbol{\delta}^{\star}_{\mathcal{Z}} = \boldsymbol{d}^{\star}_{\mathcal{Z}}/(\boldsymbol{e}^T\boldsymbol{d}^{\star}_{\mathcal{Z}})$. $\boldsymbol{\delta}^{\star}_{\mathcal{Z}}$ is the extreme point of $\bar{\Delta}_{(\mathbf{0},\boldsymbol{e})} = \left\{\boldsymbol{d}_{\mathcal{Z}} \mid \bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}}\boldsymbol{d}_{\mathcal{Z}} = 0\,,\ \boldsymbol{e}^T\boldsymbol{d}_{\mathcal{Z}} = 1\,,\ \boldsymbol{d}_{\mathcal{Z}} \geq \mathbf{0}\right\}$ that represents the same direction normalized with a different weight vector. Hence, $Supp(\boldsymbol{\delta}^{\star}_{\mathcal{Z}}) = \mathcal{S}$ and using the result of

Zaghrouti et al.,

$$\begin{aligned} \boldsymbol{d}^\star_{\mathcal{Z}} \text{ is an integral direction} &\Leftrightarrow \boldsymbol{\delta}^\star_{\mathcal{Z}} \text{ is an integral direction} \\ &\Leftrightarrow \mathit{Supp}(\boldsymbol{\delta}^\star_{\mathcal{Z}}) \text{ is column-disjoint} \\ &\Leftrightarrow \mathcal{S} \text{ is column-disjoint} \end{aligned}$$

The rest of the proposition follows from the normalization constraint $\bar{\boldsymbol{w}}^T \boldsymbol{d}_{\mathcal{Z}} = 1$. □

$\mathcal{S} = \mathit{Supp}(\boldsymbol{d}_{\mathcal{Z}})$ is the set of columns to be pivoted into the reduced basis to improve the value of the solution. By Lemma 6, all the extreme integral solutions are minimal. As shown by Zaghrouti et al. (2014), the support of $\boldsymbol{d} = \boldsymbol{T}\boldsymbol{d}_{\mathcal{Z}}$ corresponds to a nondecomposable set of Balas, which yields Corollary 8.

Corollary 8. *Given $\boldsymbol{d}^\star_{\mathcal{Z}} \in \bar{\Delta}^{int}_{\boldsymbol{w}} \cap \mathit{Ext}\left(\bar{\Delta}_{\boldsymbol{w}}\right)$, the corresponding extreme direction $\boldsymbol{d}^\star = \boldsymbol{T}\boldsymbol{d}^\star_{\mathcal{Z}}$ is*

$$d^\star_j = \begin{cases} -\alpha & \text{if } j \in \mathcal{R} \\ \alpha & \text{if } j \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{S} = \mathit{Supp}(\boldsymbol{d}^\star_{\mathcal{Z}})$, $\mathcal{R} = \mathit{Supp}(\boldsymbol{d}^\star_{\mathcal{P}})$, and $\alpha = 1/\bar{W}_{\mathcal{S}}$. Moreover, $r(\boldsymbol{d}^\star) = \bar{W}_{\mathcal{S}}$.

The next integer solution is then $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$ defined as

$$x^1_j = \begin{cases} 1 & \text{if } j \in \mathcal{S} \cup (\mathcal{P} \setminus \mathcal{R}) \\ 0 & \text{if } j \in \mathcal{R} \cup (\mathcal{Z} \setminus \mathcal{S}) \end{cases}$$

and the partition of the variables becomes (at $\boldsymbol{x}^{k+1}$): $\mathcal{P}^1 = \mathcal{S} \cup (\mathcal{P} \setminus \mathcal{R})$ and $\mathcal{Z}^1 = \mathcal{R} \cup (\mathcal{Z} \setminus \mathcal{S})$.

Classical linear programming theory ensures that at least one of the optimal solutions of R-MNA$^{\bar{\boldsymbol{w}}}$ is an extreme point of $\bar{\Delta}_{\boldsymbol{w}}$, i.e., the direction of an edge at $\boldsymbol{x}^k$. Solving the problem with the simplex algorithm naturally yields such an extreme solution. As seen in Equation (4.6), the normalization constraint influences the marginal cost of the vertices of $\bar{\Delta}_{\boldsymbol{w}}$ and, given any minimal feasible integral direction $\boldsymbol{d}$ with negative cost $\boldsymbol{c}^T\boldsymbol{d} < 0$, there exists $\boldsymbol{w} \in \mathbb{R}^n$ such that $\boldsymbol{d}^{\boldsymbol{w}} = \boldsymbol{d}/(\boldsymbol{w}^T\boldsymbol{d})$ is an optimal solution of MNA$^{\boldsymbol{w}}$ that represents the same geometric direction. From Equation (4.7), we see that a judicious choice of $\boldsymbol{w}$ can also ensure that the optimal solution $\boldsymbol{d}^\star_{\mathcal{Z}}$ of R-MNA$^{\boldsymbol{w}}$ found by the simplex algorithm leads to an integer $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + r(\boldsymbol{d}^\star_{\mathcal{Z}})\boldsymbol{d}^\star_{\mathcal{Z}}$. These observations imply that, given a solution $\boldsymbol{x}^k$, one can find a normalization weight vector $\boldsymbol{w}^k$ (that depends on $\boldsymbol{x}^k$) such that the solution of R-MNA$^{\boldsymbol{w}^k}$ furnishes an edge leading to an integer neighbor of $\boldsymbol{x}^k$. The determination of a

satisfying $\boldsymbol{w}^k$ at each iteration is hence sufficient to reach optimality in a finite number of steps. Interestingly, the following theorem shows that, given a starting solution $\boldsymbol{x}^0$, one can find a normalization vector $\boldsymbol{w}^\star$ (that depends only on $\boldsymbol{x}^0$) such that using $\boldsymbol{w}^k = \left(\boldsymbol{0}, \boldsymbol{w}^\star_{\mathcal{Z}^k}\right)$ at each iteration is sufficient for R-MNA$^{\boldsymbol{w}^k}$ to furnish an integer direction at each intermediate solution $\boldsymbol{x}^k$. Such a vector corresponds to constant reduced weights $\bar{w}_j = w^\star_j$. Moreover, one can guarantee that the resulting all-integer sequence $\left(\boldsymbol{x}^k\right)_{k\in\{1,\dots,K\}}$ ultimately leads to the optimal solution of SPP.

Theorem 9. *Let $\boldsymbol{x}^0 \in \mathcal{F}_{\text{SPP}}$. There exists $\boldsymbol{w}^\star \in \mathbb{R}^n_+$ such that the repeated solution of* MNA$^{\boldsymbol{w}^k}$*, where $\boldsymbol{w}^k = \left(\boldsymbol{0}_{\mathcal{P}^k}, \boldsymbol{w}^\star_{\mathcal{Z}^k}\right)$ (i.e., $\bar{\boldsymbol{w}}^k = \boldsymbol{w}^\star_{\mathcal{Z}^k}$), yields an augmenting sequence of (integer) solutions of* SPP *leading to an optimal solution, i.e., satisfying conditions C1–C4.*

Before proving this result, we discuss its implications. Theorem 9 does not indicate how to find the right normalization weights, but it provides a strong theoretical foundation for the method we present here. It also confirms the potential effects of the normalization on the outcome of the algorithm. To prove Theorem 9, we will need the following two lemmas. Lemma 10 shows that we can decompose the direction that links two integer solutions into minimal directions (i.e., with minimal support, as mentioned above). Lemma 11 gives a sufficient condition on the normalization weights to find the optimal solution of R-MNA.

Lemma 10. *Let $\boldsymbol{x}^0, \boldsymbol{x}^\star \in \mathcal{F}_{\text{SPP}}$, and $\mathcal{S}^\star = Supp([\boldsymbol{x}^\star - \boldsymbol{x}^0]_{\mathcal{Z}}) = \left\{j | x^\star_j - x^0_j > 0\right\}$. There exists a partition of $\mathcal{S}^\star$ into $\bigcup_{q=0}^{Q-1} \mathcal{S}^q$ such that for each $q \in \{0, \dots Q-1\}$, $\mathcal{S}^q$ is irreducible at $\boldsymbol{x}^0$. Moreover, for any permutation σ of $\{0, \dots, Q-1\}$ and with $\boldsymbol{d}^q$ being any associated direction corresponding to $\mathcal{S}^q$ (regardless of the normalization), the sequence starting at $\boldsymbol{x}^0$ defined by*

$$\boldsymbol{x}^{q+1} = \boldsymbol{x}^q + r^q \boldsymbol{d}^{\sigma(q)} \tag{4.9}$$

satisfies (1) $\boldsymbol{x}^q \in \mathcal{F}_{\text{SPP}}$, (2) $\boldsymbol{x}^q$ and $\boldsymbol{x}^{q+1}$ are neighbors in $\mathcal{F}_{\text{SPP}^{\text{RL}}}$, and (3) $\boldsymbol{x}^Q = \boldsymbol{x}^\star$.

Proof. Let $\boldsymbol{x}^0$, $\boldsymbol{x}^\star$, and $\mathcal{S}^\star$ be as defined in the statement of the lemma. If $\mathcal{S}^\star = \emptyset$ the result holds trivially. Suppose now that $\mathcal{S}^\star \neq \emptyset$.

Partition. By definition, there exists an irreducible $\mathcal{S}^0 \subseteq \mathcal{S}^\star$. Since $\boldsymbol{x}^0$ and $\boldsymbol{x}^\star$ are integer, $\mathcal{S}^\star$ is column-disjoint, hence so is $\mathcal{S}^0$. By Proposition 7, the associated direction $\boldsymbol{d}^0$ leads to an integer solution $\boldsymbol{x}^1$ that is a neighbor of $\boldsymbol{x}^0$ in $\mathcal{F}_{\text{SPP}^{\text{RL}}}$. By iterating the process until $\mathcal{S}^\star = \emptyset$, we partition $\mathcal{S}^\star$ into $\bigcup_{q=0}^{Q-1} \mathcal{S}^q$.

Rows involved. Note that, since $\mathcal{S}^\star$ is column-disjoint, the nonzero entries of the columns of $\boldsymbol{A}$ ($\boldsymbol{A}_{\cdot\mathcal{S}^0}, \dots, \boldsymbol{A}_{\cdot\mathcal{S}^{Q-1}}$) that are involved in the different directions appear in disjoint sets of

rows. Therefore, the order in which these sets are built and the order in which the directions are followed do not influence the process and thus the lemma holds. □

Lemma 11. *Let* $\boldsymbol{\epsilon} = (\mathbf{0}, \boldsymbol{e})$, $\boldsymbol{d}^{\boldsymbol{\epsilon}} \in Ext\,(\Delta_{\boldsymbol{\epsilon}})$ *be an improving feasible direction* ($\boldsymbol{c}^T\boldsymbol{d}^{\boldsymbol{\epsilon}} < 0$), *and* $\mathcal{S} = Supp(\boldsymbol{d}^{\boldsymbol{\epsilon}}_{\mathcal{Z}})$. *Let* $M \in \mathbb{R}_+$ *and* $\bar{\boldsymbol{w}} \in \mathbb{R}^{n-p}$ *be such that for all* $j \in \mathcal{Z}$, $\bar{w}_j = \mu$ *if* $j \in \mathcal{S}$, *and* $\bar{w}_j > M$ *otherwise. Let* $\boldsymbol{w} = (\mathbf{0}, \bar{\boldsymbol{w}})$ *and* $\boldsymbol{d}^{\boldsymbol{w}} = \boldsymbol{d}^{\boldsymbol{\epsilon}} / \left(\boldsymbol{c}^T\boldsymbol{d}^{\boldsymbol{\epsilon}}\right)$ *be the direction that corresponds to* $\boldsymbol{d}^{\boldsymbol{\epsilon}}$ *for the normalization weights* $\boldsymbol{w}$. *If*

$$\frac{M}{\mu} > \max\left\{ \left(\sum_{j \in \mathcal{Z}\setminus\mathcal{S}} u^{\boldsymbol{\epsilon}}_j \right)^{-1} \left(\frac{\boldsymbol{c}^T\boldsymbol{u}^{\boldsymbol{\epsilon}}}{\boldsymbol{c}^T\boldsymbol{d}^{\boldsymbol{\epsilon}}} - \sum_{j \in \mathcal{S}} u^{\boldsymbol{\epsilon}}_j \right) \mid \boldsymbol{u}^{\boldsymbol{\epsilon}} \in Ext\,(\Delta_{\boldsymbol{\epsilon}}) \setminus \{\boldsymbol{d}^{\boldsymbol{\epsilon}}\} \right\}, \tag{4.10}$$

then $\boldsymbol{d}^{\boldsymbol{w}}_{\mathcal{Z}}$ *is an optimal solution of* R-MNA$^{\boldsymbol{w}}$.

Proof. Let $\boldsymbol{d}^{\boldsymbol{\epsilon}}$, $\mathcal{S}$, M, $\bar{\boldsymbol{w}}$, $\boldsymbol{w}$, and $\boldsymbol{d}^{\boldsymbol{w}}$ be as defined in the statement of Lemma 11 and suppose that Equation 4.10 holds. First, note that since $\boldsymbol{d}^{\boldsymbol{\epsilon}} \in Ext\,(\Delta_{\boldsymbol{\epsilon}})$, direction $\boldsymbol{d}^{\boldsymbol{\epsilon}}$ is minimal (Lemma 6). Hence, for any $\boldsymbol{u}^{\boldsymbol{\epsilon}} \in Ext\,(\Delta_{\boldsymbol{\epsilon}}) \setminus \{\boldsymbol{d}^{\boldsymbol{\epsilon}}\}$, $\sum_{j \in \mathcal{Z}\setminus\mathcal{S}} \bar{u}^{\boldsymbol{\epsilon}}_j > 0$ and Equation 4.10 is well-defined. Given another feasible direction $\boldsymbol{u}^{\boldsymbol{w}} \in Ext\,(\Delta_{\boldsymbol{w}}) \setminus \{\boldsymbol{d}^{\boldsymbol{w}}\}$, let $\boldsymbol{u}^{\boldsymbol{\epsilon}} = \boldsymbol{u}^{\boldsymbol{w}} / \left(\boldsymbol{\epsilon}^T\boldsymbol{u}^{\boldsymbol{w}}\right)$ be the corresponding direction in $\Delta_{\boldsymbol{\epsilon}}$. If $\boldsymbol{c}^T\boldsymbol{u}^{\boldsymbol{w}} \geq 0$, then $\boldsymbol{u}^{\boldsymbol{w}}$ is clearly not a better solution than the negative-cost $\boldsymbol{d}^{\boldsymbol{w}}$. If $\boldsymbol{c}^T\boldsymbol{u}^{\boldsymbol{w}} < 0$ then

$$\begin{aligned} \boldsymbol{c}^T\boldsymbol{u}^{\boldsymbol{\epsilon}} = \frac{\boldsymbol{c}^T\boldsymbol{u}^{\boldsymbol{\epsilon}}}{\boldsymbol{w}^T\boldsymbol{u}^{\boldsymbol{\epsilon}}} &= \frac{\boldsymbol{c}^T\boldsymbol{u}^{\boldsymbol{\epsilon}}}{\sum_{j \in \mathcal{S}} w_j u^{\boldsymbol{\epsilon}}_j + \sum_{j \in \mathcal{Z}\setminus\mathcal{S}} w_j u^{\boldsymbol{\epsilon}}_j} \\ &\geq \frac{\boldsymbol{c}^T\boldsymbol{u}^{\boldsymbol{\epsilon}}}{\mu \sum_{j \in \mathcal{S}} u^{\boldsymbol{\epsilon}}_j + M \sum_{j \in \mathcal{Z}\setminus\mathcal{S}} u^{\boldsymbol{\epsilon}}_j} \end{aligned}$$

since $\frac{M}{\mu}$ satisfies Equation (4.10),

$$> \frac{1}{\mu} \frac{\boldsymbol{c}^T\boldsymbol{u}^{\boldsymbol{\epsilon}}}{\sum_{j \in \mathcal{S}} u^{\boldsymbol{\epsilon}}_j + \sum_{j \in \mathcal{Z}\setminus\mathcal{S}} u^{\boldsymbol{\epsilon}}_j \left(\sum_{j \in \mathcal{Z}\setminus\mathcal{S}} u^{\boldsymbol{\epsilon}}_j \right)^{-1} \left(\frac{\boldsymbol{c}^T\boldsymbol{u}^{\boldsymbol{\epsilon}}}{\boldsymbol{c}^T\boldsymbol{d}^{\boldsymbol{\epsilon}}} - \sum_{j \in \mathcal{S}} u^{\boldsymbol{\epsilon}}_j \right)}$$

by definition, $\sum_{j \in \mathcal{S}} d^{\boldsymbol{\epsilon}}_j = 1$ and $d^{\boldsymbol{\epsilon}}_j = 0$ for all $j \in \mathcal{Z} \setminus \mathcal{S}$, hence

$$> \frac{\boldsymbol{c}^T\boldsymbol{d}^{\boldsymbol{\epsilon}}}{\sum_{j \in \mathcal{S}} \mu d^{\boldsymbol{\epsilon}}_j} = \frac{\boldsymbol{c}^T\boldsymbol{d}^{\boldsymbol{\epsilon}}}{\boldsymbol{w}^T\boldsymbol{d}^{\boldsymbol{\epsilon}}} = \boldsymbol{c}^T\boldsymbol{d}^{\boldsymbol{w}}$$

and the result holds. □

Proof. (Proof of Theorem 9) Let $\boldsymbol{x}^0 \in \mathcal{F}_{\text{SPP}}$ and $\boldsymbol{x}^{\star\star}$ be an optimal solution of SPP. Let $\mathcal{S}^{\star\star} = \left\{j | x_j^{\star\star} - x_j^0 > 0\right\}$ and $\mathcal{S}^0, \ldots, \mathcal{S}^{Q-1}$ be the partition of $\mathcal{S}^{\star\star}$ described in Lemma 10. Further, let $\boldsymbol{d}^{\boldsymbol{\epsilon},0}, \ldots, \boldsymbol{d}^{\boldsymbol{\epsilon},Q-1}$ be the directions associated with these sets, normalized for $\boldsymbol{w} = \boldsymbol{\epsilon} = (\boldsymbol{0}, \boldsymbol{e})$. We have $\boldsymbol{x}^{\star\star} = \boldsymbol{x}^0 + \sum_{q=0}^{Q-1} r^{\boldsymbol{\epsilon},q} \boldsymbol{d}^{\boldsymbol{\epsilon},q}$. First, observe that all the directions satisfy $\boldsymbol{c}^T \boldsymbol{d}^{\boldsymbol{\epsilon},q} \leq 0$. Indeed, assume without loss of generality that $\boldsymbol{c}^T \boldsymbol{d}^{\boldsymbol{\epsilon},0} > 0$; then $\tilde{\boldsymbol{x}} = \boldsymbol{x}^0 + \sum_{q=1}^{Q-1} r^{\boldsymbol{\epsilon},q} \boldsymbol{d}^q$ is a solution of the SPP (by Lemma 10) of strictly better cost than the optimal $\boldsymbol{x}^{\star\star}$. Second, we can assume without loss of generality that the first K directions $\boldsymbol{d}^{\boldsymbol{\epsilon},0}, \ldots, \boldsymbol{d}^{\boldsymbol{\epsilon},K-1}$ are of strictly negative cost, while the others satisfy $\boldsymbol{c}^T \boldsymbol{d}^{\boldsymbol{\epsilon},q} = 0$ (for $q \in \{K, \ldots, Q-1\}$). Let $\boldsymbol{x}^\star = \boldsymbol{x}^0 + \sum_{k=0}^{K-1} r^{\boldsymbol{\epsilon},k} \boldsymbol{d}^{\boldsymbol{\epsilon},k}$, which is an optimal solution of SPP. Third, define $\boldsymbol{w}^\star$ as

$$w_j^\star = \begin{cases} M^k & \text{if } j \in \mathcal{S}^k \text{ for } k = 0, \ldots, K-1 \\ M^K & \text{otherwise (including } j \in \mathcal{P}^0) \end{cases}, \tag{4.11}$$

where M^k is M to the power of k and M is a sufficiently large number (as suggested in Lemma 11 where μ takes the value 1).

We will now prove by induction that the iterative solution of R-MNA with reduced weights given by $\bar{w}_j = w_j^\star$ yields directions $\boldsymbol{d}^0, \ldots, \boldsymbol{d}^{K-1}$ that are proportional to $\boldsymbol{d}^{\boldsymbol{\epsilon},0}, \ldots \boldsymbol{d}^{\boldsymbol{\epsilon},K-1}$ (and that the corresponding sequence of solutions therefore satisfies the requirements of Theorem 9). To prove this statement, we will show that all the conditions of Lemma 11 are satisfied at each step of the process.

— *Initialization*: At $\boldsymbol{x}^0$, $\bar{w}_j = M^0 = \mu$ for all $j \in \mathcal{S}^0$, and $\bar{w}_j \geq M$ otherwise. Hence, provided M is sufficiently large, $M/\mu = M$ satisfies the condition of Lemma 11 (Equation 4.10), and the optimal solution $\boldsymbol{d}^0$ of R-MNA$^{(\boldsymbol{0}, \boldsymbol{w}^\star_{\mathcal{Z}^0})}$ is proportional to $\boldsymbol{d}^{\boldsymbol{\epsilon},0}$.

— *Induction*: The induction is straightforward except for one point. Since the different directions involve disjoint supports, no column that entered the reduced basis $\mathcal{P}$ at step $k \in \{0, \ldots, K-1\}$ will leave it at a later step $k' > k$. Therefore, after k steps have been performed, for all nonbasic variables $j \in \mathcal{Z}^k$, $\bar{w}_j \geq M^k$. With the notation of Lemma 11, $\mu = M^k$ for all $j \in \mathcal{S}^k$ while all other weights are greater than M^{k+1}: $M^{k+1}/\mu = M$ remains large enough and the conditions of Lemma 11 still apply.

Hence, we have proved that using the weights $\bar{w}_j = w_j^\star$ in each of the augmentation problems R-MNA suffices to ensure that the sequence $\boldsymbol{d}^0, \ldots, \boldsymbol{d}^k$ is proportional to $\boldsymbol{d}^{\boldsymbol{\epsilon},0}, \ldots, \boldsymbol{d}^{\boldsymbol{\epsilon},Q-1}$ (since they have the same support and since they are all minimal directions, they must be proportional). By definition of these directions and according to Lemma 10, the corresponding sequence of solutions $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + r^k \boldsymbol{d}^k$ is a sequence of integer neighbors of $\mathcal{F}_{\text{SPP}^{\text{RL}}}$ (C1, C4). Moreover, as assumed in the first part of this proof, for all $k \in \{0, \ldots, K-1\}$, $\boldsymbol{c}^T \boldsymbol{d}^k < 0$,

therefore the sequence is strictly decreasing (C3). As previously mentioned, the last element $\boldsymbol{x}^\star$ of the sequence is optimal for the SPP (C2), which completes the proof. □

4.2.5 Decomposition of the Augmentation Problem

Before studying some specific weight vectors, we will discuss another feature of our algorithm: the decomposition of R-MNA$^{\boldsymbol{w}}$ into two smaller subproblems. We first introduce a definition and recall previous results.

Definition 3. Given $j \in \mathcal{Z}$, column $\boldsymbol{A}_{\cdot j}$ is said to be *compatible* with the working basis $\mathcal{P}$ if $\boldsymbol{A}_{\cdot j} \in Span(\boldsymbol{A}_{\cdot j} | j \in \mathcal{P})$. The same definition applies to index j and variable x_j. The other columns are said to be *incompatible*. $\mathcal{C}$ and $\mathcal{I}$ respectively denote the sets of all compatible and incompatible indices.

Lemma 12. (Lemma 1 of Rosat et al. (2014)) *Column $\boldsymbol{A}_{\cdot j}$ is compatible iff it is the sum of a subset of the basic columns. This set is unique and denoted $\mathcal{R}_j$, and $\boldsymbol{A}_{\cdot j} = \sum_{i \in \mathcal{R}_j} \boldsymbol{A}_{\cdot i}$.*

Observation 1. Column $\boldsymbol{A}_{\cdot j}$ is compatible iff $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}} \boldsymbol{A}_{\cdot j} = 0$.

Given $j \in \mathcal{C}$, its associated direction $\boldsymbol{\delta}^j$ is defined to be

$$\forall i \in \{1, \dots n\}, \delta_i^j = \begin{cases} -1/W^j & \text{if } i \in \mathcal{R}_j \\ 1/W^j & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

where $W^j = w_j - \sum_{i \in \mathcal{R}_j} w_j$ is feasible and integer. Also, $r(\boldsymbol{\delta}^j) = W^j$.

Lemma 13. *$\boldsymbol{\delta}^j \in Ext\left(\Delta_{\boldsymbol{w}}\right)$ and is minimal for $\Delta_{\boldsymbol{w}}$.*

Proof. By the definition of $\boldsymbol{\delta}^j$ and Observation 1, $\boldsymbol{\delta}^j \in \bar{\Delta}_{\boldsymbol{w}}$. Since $\boldsymbol{\delta}^j_{\mathcal{Z}}$ has only one nonzero entry, it is necessarily minimal in $\bar{\Delta}_{\boldsymbol{w}}$. By Lemma 6, $\boldsymbol{\delta}^j_{\mathcal{Z}} \in Ext\left(\bar{\Delta}_{\boldsymbol{w}}\right)$, so by Proposition 2, the lemma holds. □

If the reduced cost of $\boldsymbol{\delta}^j$, $\bar{c}_j = c_j - \sum_{i \in \mathcal{R}_j} c_j$, satisfies $\bar{c}_j < 0$, then $\boldsymbol{\delta}^j$ is augmenting (and so is the corresponding normalized direction). Following $\boldsymbol{\delta}^j$ is equivalent to pivoting j into the working basis and making $\mathcal{R}_j$ exit. Incompatible variables yield degenerate pivots. The following decomposition theorem justifies the use of the decomposition.

Theorem 14.

$$z^\star_{\text{R-MNA}^{\boldsymbol{w}}} = \min\left\{z^\star_{\text{RP}}, z^\star_{\text{R-MNA}^{\boldsymbol{w}}_{\mathcal{I}}}\right\} \tag{4.13}$$

where R-MNA$^{\boldsymbol{w}}_{\mathcal{I}}$ *is the restriction of* R-MNA$^{\boldsymbol{w}}$ *to the incompatible variables, and the reduced problem* RP *is*

$$z^{\star}_{\mathrm{RP}} = \min\left\{ \frac{\bar{c}_j}{W^j} | j \in \mathcal{C} \right\}, \tag{RP}$$

which is the normalized reduced-cost problem.

Proof. Let $\boldsymbol{d} \in \mathit{Ext}\,(\Delta_{\boldsymbol{w}})$, so $\boldsymbol{d}_{\mathcal{Z}} \in \mathit{Ext}\left(\bar{\Delta}_{\boldsymbol{w}}\right)$. Let $\mathcal{S} = \mathit{Supp}(\boldsymbol{d}_{\mathcal{Z}})$. We will show that either $\mathcal{S} \subseteq \mathcal{I}$ or there exists $j \in \mathcal{C}$ such that $\boldsymbol{d} = \boldsymbol{\delta}^j$. If this assertion is true, the result will hold.

Suppose that $\mathcal{S} \nsubseteq \mathcal{I}$ and let $j \in \mathcal{C}$ be such that $d_j > 0$. By Lemma 6, $\boldsymbol{d}_{\mathcal{Z}}$ is minimal in $\bar{\Delta}_{\boldsymbol{w}}$. Moreover, $\mathit{Supp}(\boldsymbol{\delta}^j_{\mathcal{Z}}) \subseteq \mathcal{S}$. Therefore, $\boldsymbol{\delta}^j_{\mathcal{Z}}$ is a feasible solution whose support is contained in the support of the minimal solution $\boldsymbol{d}_{\mathcal{Z}}$. Hence, $\boldsymbol{d}_{\mathcal{Z}} = \boldsymbol{\delta}^j_{\mathcal{Z}}$ and either $\mathcal{S} \subseteq \mathcal{I}$ or there exists $j \in \mathcal{C}$ such that $\boldsymbol{d} = \boldsymbol{\delta}^j_{\mathcal{Z}}$. □

Theorem 14 strengthens the result of Zaghrouti et al. (2014) that states that, for $\boldsymbol{w} = (\boldsymbol{0}, \boldsymbol{e})$ (i.e., $\bar{\boldsymbol{w}} = \boldsymbol{e}$), $z^{\star}_{\text{R-MNA}^{\boldsymbol{w}}} < 0 \Leftrightarrow (z^{\star}_{\mathrm{RP}} < 0$ or $z^{\star}_{\text{R-MNA}^{\boldsymbol{w}}_{\mathcal{I}}} < 0)$. Our theorem strengthens this in two ways: any normalization constraint can be used, and we have shown the equality of the left-hand and right-hand sides. This result justifies the algorithmic scheme given in the next section: first look for an improving compatible column; if none is found, look for a linear combination of incompatible columns that is compatible overall.

4.2.6 Algorithmic Scheme

Algorithm 2 summarizes the previous ideas. Various branching and cutting-plane techniques can be used to encourage integral solutions; they mostly derive from primal methods. A study of the effects of cutting planes in ISUD can be found in the work of Rosat et al. (2014) and Rosat et al. (2015b). Examples of branching techniques can be found in that of Zaghrouti et al. (2014).

4.3 Normalization Constraint in ISUD

We now study some specific normalization constraints. First, we present the constraint currently used in ISUD ($\boldsymbol{w} = (\boldsymbol{0}, \boldsymbol{e})$, $\bar{\boldsymbol{w}} = \boldsymbol{e}$) and give a new theoretical result that modifies the algorithm when this constraint is used. Second, we present the standard MMA constraint ($\boldsymbol{w} = (-\boldsymbol{e}, \boldsymbol{e})$). Third, we introduce two new constraints and explain their advantages. The normalization constraint is effectively a pricing rule: Equation (4.3) shows that this constraint directly influences the cost function.

Algorithm 2: Integral Simplex Using Decomposition with Normalization Weight Vector $\boldsymbol{w}$

Input: $\boldsymbol{x}^0$, a solution of SPP.
Output: $\boldsymbol{x}^k$, a better solution of SPP.

Compute partition $\left(\mathcal{P}^k, \mathcal{C}^k, \mathcal{I}^k\right)$ associated with $\boldsymbol{x}^k$; $k \leftarrow 0$;
while *true* **do**
  if $z^\star_{\text{RP}} < 0$ **then**
    $\boldsymbol{d}^i \leftarrow$ an optimal solution of RP ($\bar{c}_i < 0$, $i \in \mathcal{C}^k$);
    $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + r(\boldsymbol{d}^i)\boldsymbol{d}^i$; $k \leftarrow k+1$; Update $\left(\mathcal{P}^k, \mathcal{C}^k, \mathcal{I}^k\right)$;
  else if $z^\star_{\text{R-Mna}^{\boldsymbol{w}}} < 0$ **then**
    $\boldsymbol{d}^\star_{\mathcal{Z}} \leftarrow$ an optimal solution of R-Mna$^{\boldsymbol{w}}$; $\boldsymbol{d}^\star \leftarrow \boldsymbol{T}\boldsymbol{d}_{\mathcal{Z}}$;
    if $\boldsymbol{d}^\star_{\mathcal{Z}}$ *is column-disjoint* **then**
      $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$; $k \leftarrow k+1$; Update $\left(\mathcal{P}^k, \mathcal{C}^k, \mathcal{I}^k\right)$;
    else
      if *stopping criterion is reached* **then**
        return $\boldsymbol{x}^k$ (potentially nonoptimal);
      else
        Use branching or cutting-plane techniques or modify $\boldsymbol{w}$ to encourage column-disjoint solutions in R-Mna$^{\boldsymbol{w}}$;
  else
    return $\boldsymbol{x}^k$ (optimal)

4.3.1 Maximum Incoming Mean Augmentation (Mima)

To date, the ISUD algorithm has always been run with a normalization constraint on the incoming variables of $\mathcal{Z}$ (or $\mathcal{I}$ in the decomposition). That is, the algorithm solved the following augmentation, called the *maximum incoming mean augmentation* (Mima):

$$z^\star_{\text{Mima}} = \min\left\{\bar{\boldsymbol{c}}^T\boldsymbol{d}_{\mathcal{Z}} \,|\, \boldsymbol{d}_{\mathcal{Z}} \in \bar{\Delta}_{(\mathbf{0},e)}\right\} \tag{Mima}$$

i.e., the normalization constraint was $\sum_{j\in\mathcal{Z}} d_j = 1$.

Recall that, from Equation (4.3), this is equivalent to optimizing over the cost function $\frac{\boldsymbol{c}^T\boldsymbol{d}}{\boldsymbol{e}^T\boldsymbol{d}_{\mathcal{Z}}}$. Hence, the normalization constraint tends to encourage solutions whose supports $\mathcal{S} = \mathit{Supp}(\boldsymbol{d}_{\mathcal{Z}})$ have fewer variables. When there are fewer variables with nonnegative values there is a greater likelihood that the direction will be integral. Example 1 gives an illustration of this normalization constraint.

Example 1. Let us give a simple example of what an augmentation problem can be. Let

$$
\begin{array}{rl}
j = & \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\boldsymbol{A} = & \left[\begin{array}{ccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1
\end{array}\right] \\
\boldsymbol{c} = & \left[\begin{array}{ccccccc} 3 & 4 & 2 & 2 & 1 & 1 & 1 \end{array}\right] \\
\boldsymbol{d}^1 = & \left[\begin{array}{ccccccc} -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \end{array}\right] \\
\boldsymbol{d}^2 = & \left[\begin{array}{ccccccc} -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{array}\right]
\end{array}
$$

In this case, $\mathcal{S}_1 = \mathit{Supp}(\boldsymbol{d}^1_{\mathcal{Z}}) = \{3, 4\}$, $\mathcal{S}_2 = \mathit{Supp}(\boldsymbol{d}^2_{\mathcal{Z}}) = \{5, 6, 7\}$, and the corresponding exiting set for both directions is $\mathcal{R} = \{1, 2\}$. Here, with the normalization constraint $\boldsymbol{e}^T\boldsymbol{d}_{\mathcal{Z}} = 1$, direction $\boldsymbol{d}^1$ is better than $\boldsymbol{d}^2$ since $\boldsymbol{c}^T\boldsymbol{d}^1 = -\frac{3}{2} < -\frac{4}{3} = \boldsymbol{c}^T\boldsymbol{d}^2$.

Note also that following $\boldsymbol{d}^i$, $i \in \{1, 2\}$, leads to $\boldsymbol{x}^i = \boldsymbol{x}^k + r(\boldsymbol{d}^i)\boldsymbol{d}^i$. The corresponding change in the objective function is thus $\boldsymbol{c}^T(r(\boldsymbol{d}^i)\boldsymbol{d}^i) = r(\boldsymbol{d}^i)\boldsymbol{c}^T\boldsymbol{d}^i$. Since $r(\boldsymbol{d}^1) = 2$ and $r(\boldsymbol{d}^2) = 3$, the modifications are -3 for $\boldsymbol{d}^1$ and -4 for $\boldsymbol{d}^2$. The best normalized solution is not necessarily the one yielding the largest improvement after the step has been computed. Here, $\bar{\boldsymbol{w}} = \boldsymbol{e}$ tends to minimize $|\mathit{Supp}(\boldsymbol{d}_{\mathcal{Z}})|$ ($|\mathcal{S}_1| < |\mathcal{S}_2|$). However, as in any gradient-descent algorithm, only the reduced cost is available prior to making the decision, so these steps are not known in advance.

In the seminal work of Zaghrouti et al. (2014), the choice of the normalization weights (incoming mean, $\bar{\boldsymbol{w}} = \boldsymbol{e}$) relied on the following empirical observation: the fewer variables entering the basis, the greater the likelihood that they are column-disjoint, and hence that the direction is integral. Proposition 15 below gives a theoretical result that simplifies the algorithm when $\bar{\boldsymbol{w}} = \boldsymbol{e}$. For this proposition, we index sets $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{I}$ on k (the index of the solution in the augmenting sequence) since they formally depend on the current solution.

Proposition 15. *Let $\boldsymbol{x}^k$ be an integer solution of* SPP *and $\left(\mathcal{P}^k, \mathcal{C}^k, \mathcal{I}^k\right)$ the corresponding partition of the variables. Suppose that $z^\star_{\mathrm{RP}^k} \geq 0$ and let $\boldsymbol{d}^k_{\mathcal{Z}}$ be an extreme solution of* MIMA^k *that is column-disjoint and has the lowest cost among the column-disjoint solutions. Let $\boldsymbol{d}^k = \boldsymbol{T}\boldsymbol{d}^k_{\mathcal{Z}}$. One of the following is true:*

(i) $\boldsymbol{x}^k$ *is optimal for* SPP*, or* (ii) $z^\star_{\mathrm{RP}^{k+1}} \geq 0$*,*

where $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + r(\boldsymbol{d}^k)\boldsymbol{d}^k$ *and* RP^{k+1} *is the corresponding reduced problem.*

This proposition can be rephrased as: if, at step k, no compatible column has a negative reduced cost, then this is also true at step $k+1$.

Proof. Define $\boldsymbol{x}^k$, $\mathcal{P}^k$, $\mathcal{C}^k$, $\mathcal{I}^k$, $\boldsymbol{d}^k_{\mathcal{Z}}$, $\boldsymbol{d}^k$, and $\boldsymbol{x}^{k+1}$ as in the proposition and assume that $\boldsymbol{x}^k$ is not optimal for SPP. Note that $Supp(\boldsymbol{d}^k) = \mathcal{S}^k \cup \mathcal{R}^k$. We will show that $z^\star_{\mathrm{RP}^{k+1}} \geq 0$ by contradiction.

Let $j_0 \in \mathcal{P}^{k+1}$ be the index of a negative-reduced-cost compatible column at $\boldsymbol{x}^{k+1}$: $\bar{c}^{k+1}_{j_0} < 0$. As defined in Equation (4.12), $\boldsymbol{d}^{k+1} = \boldsymbol{\delta}^{j_0}$ is thus an augmenting integral direction at $\boldsymbol{x}^{k+1}$. Let $\mathcal{R}_{j_0} \subseteq \mathcal{P}^{k+1}$ be such that $\boldsymbol{A}_{\boldsymbol{\cdot} j_0} = \sum_{i \in \mathcal{R}_{j_0}} \boldsymbol{A}_{\boldsymbol{\cdot} i}$ (as defined in Lemma 12). We will distinguish three cases that correspond to the part of the partition $\left(\mathcal{P}^k, \mathcal{C}^k, \mathcal{I}^k\right)$ to which j_0 belonged.

(i) $\underline{j_0 \in \mathcal{P}^k}$. Then j_0 left the working basis between k and $k+1$, i.e., $j_0 \in \mathcal{R}^k$. Therefore, $\mathcal{R}_{j_0}$ entered the working basis at step k and $\mathcal{R}_{j_0} \subseteq \mathcal{S}^k$. Hence, $Supp(\boldsymbol{\delta}^{j_0}) \subseteq Supp(\boldsymbol{d}^k)$. Since $\boldsymbol{d}^k \in Ext\left(\Delta^k_{\boldsymbol{e}}\right)$, $\boldsymbol{d}^k = -\boldsymbol{\delta}^{j_0}$, and $\boldsymbol{x}^{k+1}$ is a worse solution than $\boldsymbol{x}^k$, which is impossible.

(ii) $\underline{j_0 \in \mathcal{C}^k}$. If j_0 was already compatible and yielded no improvement at $\boldsymbol{x}^k$ ($z^\star_{\mathrm{RP}^k} \geq 0$) it obviously also yields no improvement at $k+1$. This contradicts $\bar{c}^{k+1}_{j_0} < 0$.

(iii) $\underline{j_0 \in \mathcal{I}^k}$. j_0 switched from incompatible to compatible. Here, we will conduct a more detailed analysis of the different sets. Without loss of generality, suppose that there exists no variable apart from those of the supports of directions $\boldsymbol{d}^k$ and $\boldsymbol{d}^{k+1} = \boldsymbol{\delta}^{j_0}$.

Recall that $\boldsymbol{x}^k$, $\boldsymbol{x}^{k+1}$, and $\boldsymbol{x}^{k+2}$ are integer, so going from one to the other is equivalent to exchanging columns between $\mathcal{P}$ and $\mathcal{Z} = \mathcal{C} \cup \mathcal{I}$. We summarize the different steps in the following chart:

	$\mathcal{P}$	$\mathcal{C}$	$\mathcal{I}$	Entering $\mathcal{P}$	Exiting $\mathcal{P}$
$\boldsymbol{x}^k$:	$\mathcal{R}^k, \mathcal{R}^{k+1}_2$		$\mathcal{S}^k_1, \mathcal{S}^k_2, j_0$		
				$\mathcal{S}^k = \mathcal{S}^k_1 \cup \mathcal{S}^k_2$	$\mathcal{R}^k$
$\boldsymbol{x}^{k+1}$:	$\mathcal{S}^k_1, \mathcal{S}^k_2, \mathcal{R}^{k+1}_2$	j_0	$\mathcal{R}^k$		
				$\mathcal{S}^{k+1} = \{j_0\}$	$\mathcal{R}^{k+1} = \mathcal{S}^k_2 \cup \mathcal{R}^{k+1}_2$
$\boldsymbol{x}^{k+2}$:	$\mathcal{S}^k_1, j_0$		$\mathcal{R}^k, \mathcal{S}^k_2, \mathcal{R}^{k+1}_2$		

At $\boldsymbol{x}^k$, the augmenting direction chosen is $\boldsymbol{d}^k = \left(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\right)/|\mathcal{S}^k|$. Consider now the

feasible normalized direction that *could have been* followed and that leads directly from $\boldsymbol{x}^k$ to $\boldsymbol{x}^{k+2}$: $\boldsymbol{d}^\star = \left(\boldsymbol{x}^{k+2} - \boldsymbol{x}^k\right)/|\mathcal{S}^\star|$, where $\mathcal{S}^\star = Supp(\boldsymbol{x}^{k+2} - \boldsymbol{x}^k)$. Then,

$$\mathcal{S}^\star = \mathcal{S}_1^k \cup \{j_0\} \text{ and } \forall i \in \{1, \dots n\}, d_i^\star = \begin{cases} -1/|\mathcal{S}^\star| & \text{if } i \in \mathcal{R}^\star \\ 1/|\mathcal{S}^\star| & \text{if } i \in \mathcal{S}^\star \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{R}^\star = \mathcal{R}^k \cup \mathcal{R}_2^{k+1}$ is the set of columns that leaves the working basis when we go from $\boldsymbol{x}^k$ to $\boldsymbol{x}^{k+2}$. Let us compute the cost of this direction:

$$\begin{aligned} \bar{\boldsymbol{c}}^T \boldsymbol{d}_{\mathcal{Z}}^\star = \boldsymbol{c}^T \boldsymbol{d}^\star = \frac{\boldsymbol{c}^T(\boldsymbol{x}^{k+2} - \boldsymbol{x}^0)}{|\mathcal{S}^\star|} &= \frac{\boldsymbol{c}^T(\boldsymbol{x}^{k+2} - \boldsymbol{x}^{k+1})}{|\mathcal{S}^\star|} + \frac{\boldsymbol{c}^T(\boldsymbol{x}^{k+1} - \boldsymbol{x}^0)}{|\mathcal{S}^\star|} \\ &= \frac{|\mathcal{S}^k|}{|\mathcal{S}^\star|}\boldsymbol{c}^T\boldsymbol{d}^k + \frac{|\mathcal{S}^{k+1}|}{|\mathcal{S}^\star|}\boldsymbol{c}^T\boldsymbol{d}^{k+1} \\ &= \frac{|\mathcal{S}_1^k| + |\mathcal{S}_2^k|}{|\mathcal{S}_1^k| + 1}\boldsymbol{c}^T\boldsymbol{d}^k + \frac{1}{|\mathcal{S}_1^k| + 1}\boldsymbol{c}^T\boldsymbol{d}^{k+1} \end{aligned}$$

Note here that $\mathcal{S}_2^k \neq \emptyset$, otherwise $\mathcal{R}^k \cap \mathcal{R}^{k+1} = \emptyset$ and $j_0 \in \mathcal{C}^k$ was compatible at $\boldsymbol{x}^k$. Since $\boldsymbol{c}^T\boldsymbol{d}^k < 0$ and $\boldsymbol{c}^T\boldsymbol{d}^{k+1} < 0$, we have $\bar{\boldsymbol{c}}^T\boldsymbol{d}_{\mathcal{Z}}^\star < \boldsymbol{c}^T\boldsymbol{d}^k = \bar{\boldsymbol{c}}^T\boldsymbol{d}_{\mathcal{Z}}^k$. Hence, $\boldsymbol{d}^\star$ is a feasible normalized direction at $\boldsymbol{x}^k$ that leads to an integer solution, and which has a lower reduced cost than $\boldsymbol{d}^k$. This contradicts the definition of $\boldsymbol{d}^k$ (the most augmenting integer direction at $\boldsymbol{x}^k$), which completes the proof.

□

Therefore, in the special case of MIMA, when no augmenting compatible pivot can be found, we no longer need to consider the compatible variables. In this case, if, at some point in the execution of the algorithm, $z_{\text{RP}}^\star \geq 0$ (line 3 of Algorithm 2), RP no longer needs to be solved and lines 3–5 are never read again. In our experience, this property yields no significant improvement in the computational time. This is mainly due to the small number of compatible columns and the fact that the algorithm spends significantly less time solving RP than solving R-MNA$^{\boldsymbol{w}}$. However, it could have a greater impact on other families of problems where the proportion of compatible columns and the time spent in RP are higher.

4.3.2 Maximum Mean Augmentation (MMA)

Another classical normalization constraint corresponds to the *maximum mean augmentation* (MMA) problem: $\boldsymbol{w} = (-\boldsymbol{e}, \boldsymbol{e})$. We will not present a full analysis of this constraint since it has already been studied (Spille & Weismantel, 2005; Schulz & Weismantel, 2002). The

constraint is $-\sum_{j\in\mathcal{P}} d_j + \sum_{j\in\mathcal{Z}} d_j = \|\boldsymbol{d}\|_1 = 1$, and the objective function corresponds to $\left(-\boldsymbol{e}^T\boldsymbol{d}_{\mathcal{P}} + \boldsymbol{e}^T\boldsymbol{d}_{\mathcal{Z}}\right) / \|\boldsymbol{d}\|_1$. The MMA program is

$$z^\star_{\text{MIMA}} = \min\left\{\bar{\boldsymbol{c}}^T\boldsymbol{d}_{\mathcal{Z}} \,|\, \boldsymbol{d}_{\mathcal{Z}} \in \bar{\Delta}_{(-\boldsymbol{e},\boldsymbol{e})}\right\}. \tag{MMA}$$

This problem originally comes from the maximum mean cycle cancellation algorithm proposed by Goldberg and Tarjan for the minimum-cost flow problem in a directed network (see Goldberg & Tarjan (1989) for more details). Its canonical extension to generic 0–1 programs is a classical example of augmentation subproblems. For more details on its properties and its influence on the behavior of a primal algorithm see the review of Spille & Weismantel (2005). They show that this normalization constraint guarantees a pseudo-polynomial bound on the number of directions that must be found to reach optimality. Note also that in the particular case of the SPP (and of 0–1 programs in general), MMA is equivalent to the *maximum ratio augmentation* for which each term w_j of the normalization constraint is computed as the distance from its current value x_j to its furthest bound (see Schulz & Weismantel (2002) for more details).

Example 2. Let us continue with Example 1, but with the normalization constraint $\sum_j \gamma_j d_j$, $\gamma_j = -1$ if $j \in \{1,2\}$ and $+1$ if $j \in \{3,4,5,6,7\}$. We have

$$\boldsymbol{d}^1 = \begin{bmatrix} -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{d}^2 = \begin{bmatrix} -\frac{1}{5} & -\frac{1}{5} & 0 & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix}$$

and $\boldsymbol{c}^T\boldsymbol{d}^1 = -3/4 > -4/5 = \boldsymbol{c}^T\boldsymbol{d}^2$. Hence, this normalization constraint would select $\boldsymbol{d}^2$ rather than $\boldsymbol{d}^1$, unlike MIMA.

The weight vector $(-\boldsymbol{e}, \boldsymbol{e})$ measures the number of columns in $Supp(\boldsymbol{d})$, whereas $\boldsymbol{w} = \boldsymbol{\epsilon} = (\boldsymbol{0}, \boldsymbol{e})$ (i.e., $\bar{\boldsymbol{w}} = \boldsymbol{e}$) measures the cardinality of $Supp(\boldsymbol{d}_{\mathcal{Z}})$.

4.3.3 Encouraging Integrality via the Normalization Constraint

This section is a first attempt to propose new ways of encouraging integrality through specific normalization weights. We do not give an exhaustive description of the possibilities of this method. We indicate the kind of constraints that can be used to encourage integral solutions and give a foretaste of their computational influence on ISUD (Section 4.4). As mentioned in the Introduction, a follow-up paper will investigate in more detail the ways to

influence integrality and will include more computational results.

We consider the situation where a set of leaving variables $\mathcal{R}$ can be exchanged with two sets of entering variables $\mathcal{S}_1$ and $\mathcal{S}_2$. Furthermore, we assume that $\mathcal{S}_1$ is column-disjoint and $\mathcal{S}_2$ is not. The corresponding directions are denoted $\boldsymbol{d}^1$ and $\boldsymbol{d}^2$. We want to define a constraint $\boldsymbol{w}^T\boldsymbol{d} = 1$ such that R-MNA$^{\boldsymbol{w}}$ tends to select $\mathcal{S}_1$ rather than $\mathcal{S}_2$. To do this, we use parameters that allow us to encourage column-disjoint solutions, namely the norm and the incompatibility degree of a column.

Column Norm

Definition 4. The *norm* of a column $\boldsymbol{A}_{\cdot j}$ is $n_j = \|\boldsymbol{A}_{\cdot j}\|_1 = \sum_{i\in\{1,\dots m\}} |A_{ij}|$. Since $\boldsymbol{A}$ is binary, the norm of a column is just the number of its nonzero components. Moreover, for any set $\mathcal{X}$ of columns, $N_{\mathcal{X}} = \sum_{j\in\mathcal{X}} n_j$.

Given a feasible direction $\boldsymbol{d}$ and the corresponding set of entering variables $\mathcal{S} = \mathit{Supp}(\boldsymbol{d}_{\mathcal{Z}})$, $N_{\mathcal{S}} \geq N_{\mathcal{R}}$. Moreover, $N_{\mathcal{S}} = N_{\mathcal{R}}$ if and only if $\mathcal{S}$ is column-disjoint. Thus, the sum of the norms of the columns of $\mathit{Supp}(\boldsymbol{d})$ tends to be smaller for disjoint combinations. This suggests the idea of using the norms of the columns as normalization weights: $w_j = n_j$, $j \in \mathcal{Z}$ tends to encourage combinations of disjoint columns.

For instance, consider the above example of two possible directions $\boldsymbol{d}^1$ and $\boldsymbol{d}^2$ such that $\mathcal{R} = \mathit{Supp}(\boldsymbol{d}^1_{\mathcal{P}}) = \mathit{Supp}(\boldsymbol{d}^2_{\mathcal{P}})$. Suppose that these directions lead to two adjacent vertices $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ with the same cost $\boldsymbol{c}^T\boldsymbol{x}^1 = \boldsymbol{c}^T\boldsymbol{x}^2$. Since $\mathcal{S}^1$ is column-disjoint, but $\mathcal{S}^2$ is not, the corresponding reduced costs are

$$\boldsymbol{c}^T\boldsymbol{d}^1 = \frac{\boldsymbol{c}^T\left(\boldsymbol{x}^1 - \boldsymbol{x}^k\right)}{N_{\mathcal{S}^1}} < \frac{\boldsymbol{c}^T\left(\boldsymbol{x}^2 - \boldsymbol{x}^k\right)}{N_{\mathcal{S}^2}} = \boldsymbol{c}^T\boldsymbol{d}^2,$$

and the algorithm selects the column-disjoint $\boldsymbol{d}^1_{\mathcal{Z}}$ rather than the non-column-disjoint $\boldsymbol{d}^2_{\mathcal{Z}}$.

Incompatibility Degree

Definition 5. The *incompatibility degree* For $j \in \mathcal{Z}$, ι_j of a column $\boldsymbol{A}_{\cdot j}$ is the number of nonzero components of $\bar{\boldsymbol{A}}_{\cdot j}$, where we recall that $\bar{\boldsymbol{A}} = \boldsymbol{A}_{\bar{\mathcal{P}}\cdot}\boldsymbol{T} = -\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{Z}} + \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{Z}}$ (see page 37).

Note that compatible columns as defined in Definition 3 are exactly the columns with incompatibility degree $\iota_j = 0$. In the same way as the norm, the incompatibility degree measures the integrality of a solution.

Proposition 16. *If the entering set $\mathcal{S}$ is column-disjoint, then each nonzero row of $\bar{\boldsymbol{A}}_{\mathcal{Z}\mathcal{S}}$ has exactly one component equal to* 1 *and one component equal to* -1*, all others being zero.*

To prove this proposition, we need the following lemma.

Lemma 17. *Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two binary matrices. Then:*

(i) $\boldsymbol{X}$ is column-disjoint $\Leftrightarrow \boldsymbol{X}^T\boldsymbol{X}$ is diagonal;

(ii) $\boldsymbol{X}$ and $\boldsymbol{Y}$ are binary and column-disjoint $\Rightarrow \boldsymbol{XY}$ is column-disjoint.

Proof. We will prove these two points separately.

(i) follows from the definition of column-disjoint and the nonnegativity of the entries of $\boldsymbol{A}$.

(ii) Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two column-disjoint binary matrices. By (i), $\boldsymbol{D} = \boldsymbol{X}^T\boldsymbol{X}$ is diagonal, so $(\boldsymbol{XY})^T(\boldsymbol{XY}) = \boldsymbol{Y}^T\boldsymbol{DY}$. Therefore, $\left[\boldsymbol{Y}^T\boldsymbol{DY}\right]_{ij} = \sum_k (D_{kk}Y_{ki}Y_{kj})$. According to the definition of column-disjoint, if $i \neq j$, for all k, $Y_{ki}Y_{kj} = 0$, so $(\boldsymbol{XY})^T(\boldsymbol{XY})$ is diagonal. Since $\boldsymbol{X}$ and $\boldsymbol{Y}$ are binary, by (i), $\boldsymbol{XY}$ is column-disjoint.

□

Proof. (Proof of Proposition 16) Let $\boldsymbol{d} \in \mathit{Ext}\,(\Delta_{\boldsymbol{w}})$ be an extreme feasible direction at $\boldsymbol{x}^k$ and $\mathcal{S} = \mathit{Supp}(\boldsymbol{d}_{\mathcal{Z}})$ the set of entering columns. From the definition of $\bar{\boldsymbol{A}}$ (page 37), we have $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{S}} = -\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{S}} + \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{S}}$. Since by assumption $\mathcal{S}$ is column-disjoint, $\boldsymbol{A}_{\mathcal{P}\mathcal{S}}$ and $\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{S}}$ are column-disjoint. Note that the condition $\boldsymbol{Ax} = \boldsymbol{e}$ implies that every row of $\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}$ contains one and only one positive component equal to 1, all other components being zero. Hence, $\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}$ is also column-disjoint.

By Lemma 17, $\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{S}}$ is column-disjoint. Consequently, since $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{S}} = -\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{S}} + \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{S}}$, each row of $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{S}}$ has at most two nonzero components. Furthermore, by Corollary 8, the linear constraints $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{S}}\boldsymbol{d}_{\mathcal{N}} = \boldsymbol{0}$ yield $\sum_{j\in\mathcal{S}} \bar{\boldsymbol{A}}_{\bar{\mathcal{P}}j} = 0$. Hence, each nonzero row of $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{S}}$ must have exactly two nonzero components: 1 and -1. □

Proposition 16 shows that fractional solutions may have several nonzero components in every row of the matrix $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{S}}$, while integer solutions have only two. $\sum_{j\in\mathcal{S}} \iota_j$ is the number of nonzero components in the matrix $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{S}}$. Hence, like $\sum_{j\in\mathcal{S}} n_j$, the quantity $\sum_{j\in\mathcal{S}} \iota_j$ tends to be smaller for integer solutions. These properties naturally lead to the idea of using $\bar{w}_j = \iota_j$ (or $\bar{w}_j = \iota_j|\bar{c}_j|$) as normalization weights.

4.4 Numerical Results

The results [1] presented in this section establish a proof of concept for our theoretical results: we show that a judicious choice of the normalization constraint plays an important role in encouraging integral solutions in ISUD.

4.4.1 Methodology

Instances The instances presented here are those used by Zaghrouti et al. (2014) in their seminal paper on ISUD: SPPAA01, VCS1200, and VCS1600. SPPAA01 is a small flight assignment problem (823 constraints, 8,904 variables) with an average of 9 nonzero entries per column. VCS1200 is a medium-size bus driver scheduling problem (1200 constraints, 133,000 variables), and VCS1600 is a large vehicle crew scheduling problem (1,600 constraints, 571,000 variables); both have an average of 40 nonzeros per column. These numbers of nonzeros are typical of aircrew and bus driver scheduling problems and do not vary with the number of constraints. The nonzeros correspond to the number of tasks per duty. The optimal solutions of the problems are known: CPLEX [2] has solved SPPAA01, and GENCOL [3] has solved VCS1200 and VCS1600. Note that within a time limit of 10 hours, CPLEX cannot find a feasible solution for VCS1200 or VCS1600.

Initial solutions In scheduling applications, a column generally represents a sequence of tasks performed by an employee. Define the *primal information* of a solution as the percentage of consecutive tasks in that solution that are also consecutive in the optimal schedule. In practice, this quantity is not known precisely a priori, but the following two observations hold. First, crew do not often change vehicles during their duties, and their schedules therefore have many consecutive tasks in common with those of the vehicle routes. Second, when the schedule is updated, e.g., because of unforeseen events, the reoptimized schedule usually has many pieces in common with the original schedule (companies generally add penalties in the objective function to discourage changes). Thus, many consecutive tasks from the initial *paths* (vehicle routes or the original schedule) remain consecutive in the optimal schedule. In bus driver scheduling problems, the initial solution that follows the bus routes typically contains 90% of primal information (VCS1200, VCS1600). In aircrew scheduling, the figure is generally around 75% (Zaghrouti et al., 2014) (SPPAA01).

1. All the tests were performed on a Linux PC with 8 processors of 3.4 GHz.
2. The version used here is IBM CPLEX 12.5.
3. GENCOL is commercial software developed at the GERAD research center and now owned by the AD OPT company, a division of KRONOS.

Therefore, we perturb the (known) optimal solutions to generate initial solutions that contain a similar level of primal information to that seen in practice. These solutions are generally infeasible, so we add artificial columns with a high cost, as is done for the schedules that follow the bus/airplane routes. In our perturbation method, the input parameter π is the percentage of columns of the optimal solution that will appear in the new initial solution. For SPPAA01, we generated initial solutions for $\pi = 10\%$, 15%, 20%, and 35%; for VCS1200 and VCS1600, we used $\pi = 20\%$, 35%, and 50%. These parameters were chosen so that the resulting primal information (given in Table 4.1) is consistent with the typical values. The initial gaps range from 50% to 80%, depending on the instance.

Normalization weights The tests were conducted for the following normalization weights:

- MIMA, where $\bar{w}_j = 1$ for all $j \in \mathcal{Z}$;
- MMA, where $w_j = -1$ if $j \in \mathcal{P}$, 1 if $j \in \mathcal{Z}$;
- NORM, where $\bar{w}_j = n_j$ for all $j \in \mathcal{Z}$;
- DEG, where $\bar{w}_j = \iota_j$ for all $j \in \mathcal{Z}$.

Combinations or modifications of these weights could of course be tested; however, for this first study, we present only these four possibilities.

Branching technique In the example shown here, a simple nonexhaustive branching technique is used (line 14 of Algorithm 2). Whenever a solution $\boldsymbol{d}^\star$ is not column-disjoint, all the variables of $\mathcal{S}^\star = \mathit{Supp}(\boldsymbol{d}^\star_{\mathcal{Z}})$ are set to zero in R-MNA$^{\boldsymbol{w}}$ until an augmentation is performed. No backtracking is performed in this *depth-only* branching strategy (hence it is nonexhaustive). In practice, this gives good results because there is a high probability that the direction found when solving R-MNA$^{\boldsymbol{w}}$ will be column-disjoint. After we fix all the variables of the current combination to zero, the success probability is still high, and the algorithm usually finds an integer direction with this strategy.

Table 4.1 Percentage of primal information in the initial solutions of the benchmark.

			Primal Information				
Instance	m	n	$\pi = 10\%$	$\pi = 15\%$	$\pi = 20\%$	$\pi = 35\%$	$\pi = 50\%$
SPPAA01	803	8,904	71.5	75.6	80.0	86.2	-
VCS1200	1,200	133,000	-	-	87.6	91.1	93.9
VCS1600	1,600	570,000	-	-	86.8	90.9	93.9

4.4.2 Results

Table 4.2 gives the results for SPPAA01. For each perturbation parameter $\pi \in \{10, 15, 20, 25\}$, 10 different instances (initial solutions and corresponding artificial columns) are generated. Tables 4.3 and 4.4 give the results for VCS1200 and VCS1600. For each of these instances, 10 different initial solutions were generated for each $\pi \in \{20, 35, 50\}$. Column $\boldsymbol{w}$ indicates the normalization weights used; BEST is the best of the four on each instance, i.e., the one with the smallest gap, and in the event of a tie, the shortest computational time to reach the best solution. The first columns give the quality of the solution, showing how many instances of the 10 were solved to optimality (0%), with a gap $\leq 2\%$, and with a gap $> 2\%$; the mean gap is given in column MEAN. The next columns give the total computational time (ISUD), the time to reach the best solution found (BEST), and the average time per augmentation, i.e., from $\boldsymbol{x}^k$ to $\boldsymbol{x}^{k+1}$ (**AUG**). The final columns give the mean number of augmentations K performed to reach the best solution, the percentage of directions that were disjoint even before any variable was fixed (DISJ), and the mean cardinality of $\mathcal{S} = \mathit{Supp}(\boldsymbol{d}_{\mathcal{Z}})$ for disjoint ($|\mathcal{S}|^D$) and nondisjoint ($|\mathcal{S}|^{ND}$) directions. Note that, while the other mean values are computed over all executions of the algorithm, the mean number of augmentations K is based on the instances for which the final gap is $\leq 2\%$. Large gaps often mean a much smaller number of iterations, and taking these instances into account would distort the mean.

Almost all the instances were solved to optimality by at least one of the algorithms (94/100), but no algorithm is perfect.

Table 4.2 Evaluation of ISUD on the aircrew scheduling instance SPPAA01.

		GAP				Time (s)			**AUG**			
π	$\boldsymbol{w}$	0%	$\leq 2\%$	$> 2\%$	MEAN	ISUD	Best	**AUG**	K	DISJ	$\lvert\mathcal{S}\rvert^D$	$\lvert\mathcal{S}\rvert^{ND}$
10%	MIMA	3	3	4	141.3%	11.4	8.7	0.27	38	41%	4.4	112
	MMA	4	3	3	73.8%	11.8	9.4	0.26	38	46%	4.5	95
	NORM	2	3	5	87.3%	13.8	10.7	0.29	41	34%	4.5	78
	DEG	7	2	1	31.1%	12.5	10.1	0.26	39	47%	4.6	78
	BEST	**8**	**2**	**0**	**0.1%**	**12.3**	**10.3**	-	-	-	-	-
15%	MIMA	2	4	4	30.9%	15	12.2	0.30	44	36%	4	87
	MMA	8	1	1	4.4%	12.6	10	0.24	42	44%	4	69
	NORM	7	0	3	72.1%	11.5	8.5	0.23	38	42%	3.8	97
	DEG	8	1	1	29.2%	11.5	9.1	0.21	43	53%	3.9	74
	BEST	**10**	**0**	**0**	**0%**	**10.6**	**8**	-	-	-	-	-
20%	MIMA	7	2	1	20.1%	9.9	7	0.17	42	51%	3.3	67
	MMA	6	2	2	20.9%	9.6	6.4	0.16	40	52%	3.4	77
	NORM	7	3	0	0.1%	10.3	7.8	0.18	42	49%	3.5	50
	DEG	7	2	1	3.5%	10.8	7.9	0.17	45	54%	3.4	53
	BEST	**9**	**1**	**0**	**0%**	**9.4**	**7**	-	-	-	-	-
35%	MIMA	9	1	0	0%	7	4.4	0.13	34	56%	2.9	60
	MMA	9	1	0	0%	7.1	4.5	0.13	35	56%	2.9	57
	NORM	10	0	0	0%	7.4	4.5	0.13	35	55%	2.9	58
	DEG	8	2	0	0.1%	7.4	4.6	0.13	35	55%	2.9	46
	BEST	**10**	**0**	**0**	**0%**	**7.1**	**4.4**	-	-	-	-	-

Table 4.3 Evaluation of ISUD on the bus driver scheduling instance VCS1200.

		GAP				Time (s)			**AUG**			
π	$\boldsymbol{w}$	0%	$\leq 2\%$	$> 2\%$	MEAN	ISUD	Best	**AUG**	K	DISJ	$\lvert\mathcal{S}\rvert^D$	$\lvert\mathcal{S}\rvert^{ND}$
20%	MIMA	8	0	2	9.7%	93	58	3	21	56%	3	296
	MMA	9	0	1	5.5%	99	65	3.3	21	57%	3	281
	NORM	8	0	2	9.7%	87	55	2.9	21	56%	3	300
	DEG	8	0	2	9.7%	96	61	3.1	22	57%	3	289
	BEST	**9**	**0**	**1**	**5.5%**	**93**	**62**	-	-	-	-	-
35%	MIMA	9	0	1	1.2%	86	46	2.5	19	52%	2	248
	MMA	9	0	1	1.2%	87	47	2.6	18	52%	2	250
	NORM	8	0	2	3.5%	79	40	2.3	18	50%	2	280
	DEG	9	0	1	1.2%	87	47	2.6	19	52%	2	250
	BEST	**9**	**0**	**1**	**1.2%**	**83**	**43**	-	-	-	-	-
50%	MIMA	10	0	0	0%	71	28	1.9	15	48%	2	233
	MMA	10	0	0	0%	72	28	1.9	15	48%	2	232
	NORM	10	0	0	0%	69	27	1.8	15	49%	2	226
	DEG	10	0	0	0%	72	28	1.9	15	49%	2	233
	BEST	**10**	**0**	**0**	**0%**	**70**	**27**	-	-	-	-	-

Table 4.4 Evaluation of ISUD on the bus driver scheduling instance VCS1600.

		GAP				Time (s)			**AUG**			
π	$\boldsymbol{w}$	0%	$\leq 2\%$	$> 2\%$	MEAN	ISUD	Best	**AUG**	K	DISJ	$\lvert\mathcal{S}\rvert^D$	$\lvert\mathcal{S}\rvert^{ND}$
20%	MIMA	10	0	0	0%	1278	450	17.1	26	69%	3	463
	MMA	10	0	0	0%	1238	451	17	27	69%	3	483
	NORM	9	0	1	3.7%	892	395	15.5	26	66%	3	546
	DEG	10	0	0	0%	1130	469	17.3	27	70%	3	467
	BEST	**10**	**0**	**0**	**0%**	**1045**	**414**	-	-	-	-	-
35%	MIMA	10	0	0	0%	1232	312	12.9	24	68%	2	441
	MMA	10	0	0	0%	1159	304	12.6	24	68%	2	446
	NORM	9	0	1	1.1%	866	261	10.9	24	68%	2	468
	DEG	10	0	0	0%	1009	311	12.7	24	69%	2	430
	BEST	**10**	**0**	**0**	**0%**	**924**	**277**	-	-	-	-	-
50%	MIMA	9	0	1	1.1%	1084	189	11	17	60%	2	485
	MMA	9	0	1	1.1%	1048	189	11	17	60%	2	488
	NORM	9	0	1	1.4%	756	159	9.2	17	61%	2	507
	DEG	9	0	1	1.1%	868	182	10.6	17	60%	2	477
	BEST	**9**	**0**	**1**	**1.1%**	**767**	**168**	-	-	-	-	-

On SPPAA01 (Table 4.2), the performances of the various normalization weights vary, and some get much worse when the initial primal information decreases ($\pi = 10\%$). However, the best solution over the different normalization constraints is always within 2% of the optimal solution, and it is nearly always optimal. The computational times are similar for the different normalization weights, as are the time to reach the best solution (BEST) and the mean time per augmentation (**AUG**). The same holds for the overall number of iterations K and the size of the disjoint combinations $|\mathcal{S}|^D$. However, when the algorithm performs well, the percentage of disjoint solutions (DISJ) is higher and the size of the nondisjoint supports $|\mathcal{S}|^{ND}$ tends to be significantly smaller than in the other cases. For instance, when $\pi = 20\%$, the best weights are NORM and DEG, for which the mean sizes of nondisjoint combinations are respectively 50 and 53, against 67 and 77 for the other constraints. This is a consequence of the branching technique used: fixing more variables to zero increases the

risk of finding no improving feasible solutions (nonexhaustive branching).

The results for VCS1200 and VCS1600 are given in Tables 4.3 and 4.4; the conclusions that can be drawn from these tables are quite similar. First, ISUD performs well on the instances that CPLEX cannot solve. Nearly all the instances are solved to optimality, and when ISUD fails, it generally fails regardless of the normalization weights (the best-of-four also fails). Using the column norm (NORM) as the normalization vector is significantly faster but yields slightly worse results: only 53 instances were solved to optimality, against at least 56 for the other weights. Note that the mean computational time is not biased by the lower performance since the mean time is computed only for the instances for which the best solution is within 2% of the optimum. Similarly to the case of SPPAA01, the algorithm performs better but more slowly when the size of the nondisjoint combinations is smaller. All the other parameters do not vary significantly for the different normalizations.

Since the performances of the four normalizations were similar for VCS1200 and VCS1600, we drew the performance diagram of the five weights (including BEST) only for instance SPPAA01; see Figure 4.4. The instances considered are the 40 instances generated from SPPAA01 (10 for each $\pi \in \{10, 15, 20, 35\}$). The figure shows the percentage of instances solved to within 2% of the optimal solution against the computational time. This diagram confirms that no weight vector is significantly better than any other. Interestingly, the new weights introduced in this paper outperform those used until now (MIMA). This shows that the choice of the normalization vector has the potential to encourage integral solutions in ISUD.

4.5 Conclusions

We have studied the potential of the normalization-weight vector to encourage integral directions in ISUD. We have strengthened and extended the theoretical base laid by Zaghrouti et al. (2014), and we have underlined the bijection between the original and the reduced search spaces. We have proved an algorithmic improvement for the MIMA constraint currently used in ISUD. We compared it to the classical MMA and to two other normalization constraints. We have presented the theoretical background for these new constraints, and we ran numerical tests showing that the choice of the normalization weights is crucial in the performance of the algorithm. Our new constraints outperform the existing one, and nearly all the instances were solved to optimality or quasi-optimality (GAP $\leq 2\%$).

A future paper will present more normalization weights and apply them to a larger benchmark. We also intend to develop an algorithm that dynamically updates the normalization

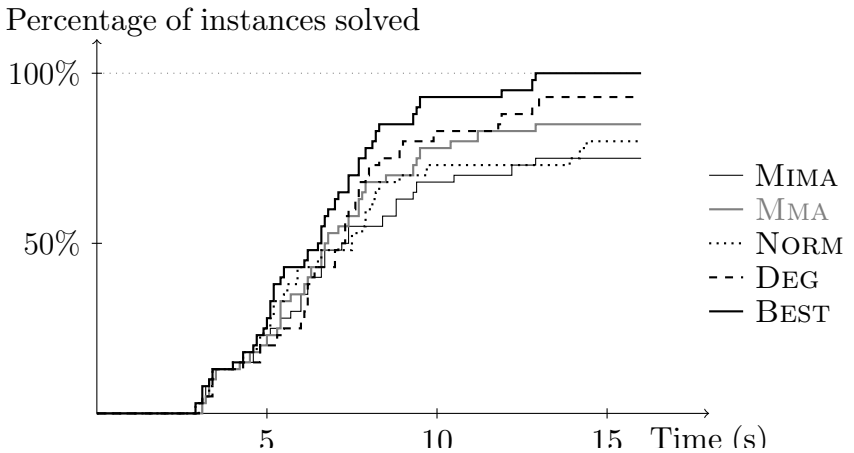


Figure 4.4 Performance diagram of ISUD on SPPAA01. An instance is considered solved if the gap is $\leq 2\%$

weights. The updates may be based on the current solution. Alternatively, we may perform updates when the augmentation problem fails to find an integral direction, to encourage integral solutions and to replace or support the current nonexhaustive branching.

Acknowledgements

This work was supported by a RD COOP research grant from CRSNG and Kronos Inc. The authors thank the two anonymous reviewers for their valuable comments that improved the quality of this work.

CHAPITRE 5 ARTICLE 2: INTEGRAL SIMPLEX USING DECOMPOSITION WITH PRIMAL CUTTING PLANES

Le texte de ce chapitre est celui de l'article *Integral simplex using decomposition with primal cutting planes* publié dans les cahiers du GERAD et soumis à *Mathematical Programming Series A and B* (Rosat et al., 2015b). Auteurs : Samuel Rosat, Issmail Elhallaoui, François Soumis, Andrea Lodi. Un résumé étendu de ce travail a été publié dans *Experimental Algorithms* (Rosat et al., 2014).

Abstract

We propose a primal algorithm for the Set Partitioning Problem based on the Integral Simplex Using Decomposition of Zaghrouti et al. (2014). We present the algorithm in a pure primal form, and relate it to other augmenting methods. We show that cutting planes can be transferred to the complementary problem, and we characterize the set of transferable cuts as a nonempty subset of primal cuts that are tight to the current solution. We prove that these cutting planes always exist, we propose efficient separation procedures for primal clique and odd-cycle cuts, and prove that their search space can be restricted to a small subset of the variables making the computation efficient. Numerical results demonstrate the effectiveness of adding cutting planes to the algorithm; tests are performed on small and large-scale set partitioning problems from aircrew and bus-driver scheduling instances up to 1,600 constraints and 570,000 variables.

Keywords 0/1 Programming, Integral Simplex, Primal Algorithms, Set-Partitioning, Primal Cutting-planes, Scheduling.

5.1 Introduction

Introduced in 1969 by Garfinkel & Nemhauser (1969), the Set Partitioning Problem (SPP) is a well known model of integer linear programming. Its popularity mainly comes from its simple expression, and the wide range of its applications. It can be expressed as [1]

SPP	Given a set $\mathcal{X}$ and a set of its subsets, $\mathcal{X}_1, \ldots, \mathcal{X}_n \subseteq \mathcal{X}$ of respective cost $c_1, \ldots, c_n$, determine a partition of $\mathcal{X}$, using only some of the $\mathcal{X}_i$, $1 \leq i \leq n$, and of minimum (or maximum) cost.

Applications range from aircrew scheduling (Desrosiers et al., 1995) to vehicle routing (Baldacci & Mingozzi, 2009) and electricity production planning (Rozenknop et al., 2013),

1. A formulation of SPP as a mathematical program is given in Section 5.2.3.

among others.

The most common method for integer linear programming, and thus SPP, is the *branch and bound*, originally introduced by Dakin (1965). That method is based on the recursive exploration of a branching tree to determine an optimal solution. Even though pruning some branches may be possible by computing lower/upper bounds with the linear relaxation, and even though standard cutting planes can tighten this linear relaxation and speed up the process (*branch and cut*), the size of the branching tree is exponential in the number of variables and the algorithm can get very slow on large instances. Moreover, in many practical cases, the branch and bound is not even able to take efficiently advantage of available initial solutions. These observations motivate the search for other methods than branch and bound to solve integer linear programs. We are obviously neither the firsts, nor probably the lasts, to raise that issue. Alternative methods based on linear programming do indeed exist, and among them, *primal algorithms*. These algorithms, sometimes described as *augmenting methods* or *all-integer* algorithms, are based on the following pattern: given a starting feasible solution, improve it iteratively to obtain a sequence of improving feasible solutions until optimality is reached. Two of their key features are to avoid the exploration of the aforementioned branching tree and take advantage of existing starting solutions. Moreover, classical methods from integer programming can be adapted to the primal framework, provided some theoretical adjustments are taken; in this work, we particularly focus on the case of cutting planes.

In the case of {0,1}-programming, all integer points of the linear relaxation of the {0,1}-program are extreme points of its linear relaxation. Therefore, one can design an all-integer algorithm such that all improvements are obtained by performing simplex pivots. Such algorithms are hence named *integral simplex* algorithms. One of the main drawbacks of algorithms based on simplex pivots is their inability to perform well on degenerate problems. In mathematical programming, degeneracy occurs when some basic variables are at one of their bounds, which is common in the particular case of SPP. In this case, it is very much likely that the value of variable entering the simplex basis cannot be modified without making the current solution infeasible. The resulting degenerate pivot leads to no change in the solution, and no improvement in the objective value. Recently, Zaghrouti et al. (2014) proposed a new algorithm for SPP, the *Integral Simplex Using Decomposition* (ISUD) which is an offspring of recent works conducted around the *Improved Primal Simplex* by Metrane et al. (2010), Elhallaoui et al. (2011), Towhidi et al. (2014), and Omer et al. (2015). It is therefore designed to take advantage of degeneracy, rather than suffer from it. Combined with (i) the canonically degenerate nature of the SPP, particularly in the industrial applications, and (ii) the observation that primal algorithms experience troubles with degeneracy,

particularly when primal cutting planes are used (Letchford & Lodi, 2003b), the previous observations on the advantageous way that ISUD copes with degeneracy show its potential to embody the next generation of primal algorithms. Furthermore, it seems natural to apply primal cutting planes techniques within an "anti-degeneracy" framework since degeneracy is reported as the main trouble experienced when adding cuts in primal methods.

From the applicative point of view, and as shown in the last part of this paper, ISUD proves highly efficient for *reoptimization*. This key feature makes it a very promising method in two particularly important cases which are reoptimization of existing solutions, and all-integer column generation. First, the need to quickly reoptimize an existing solution (a schedule) in case of unforeseen events is fundamental in many practical cases. Take for instance the example of airline crew scheduling (Stojković et al., 1998): The manpower planned schedule must often be modified to react to day-to-day operational constraints such as schedule disruptions, aircraft substitutions, crew absences, strikes or even volcanic eruptions! Second, consider the all-integer column generation process. Column generation is an optimization technique used to solve very large problems. When solving problems with integer variables, it is usually embedded in a branch-and-price framework. As for standard integer programming, the exploration of the branching tree can turn to be a very long process. In the same way that primal algorithms are an alternative to branch and bound, all-integer column generation is an alternative to branch and price. In all-integer column generation, a subset of the columns of the constraint matrix is generated in the beginning, and the optimal solution to this program, called *restricted master problem*, is found. Then, subproblems generate new columns to be appended to the matrix, and the new optimal solution for the extended matrix must be determined. One then iterates the process until no column can be generated, that improves the solution. Obviously, solving the extended problem from scratch (*branch and bound*) results in a loss of information and a probable lack of efficiency, while using the previous solution as a warm-start could give the algorithm a substantial advantage (*primal algorithms*). Hence, any efficient all-integer column generation code requires a good reoptimization method, since the global process is based on successive updates of an integral solution. Thus, ISUD is an excellent candidate for the reoptimization of the optimal solution of the restricted master problem every time it is extended.

This paper is organized as follows. A literature review on primal algorithms, primal cutting planes and integral simplex methods, as well as some notation, problem definitions and contribution statement are given in Section 5.2. The ISUD algorithm is presented in an innovative way, and new theoretical results are discussed in Section 5.3. Primal cutting planes are discussed in Section 5.4, new separation procedures are described, and we show that

the search space of the separation can be restricted without preventing from finding these cutting planes. Numerical results are displayed in Section 5.5, which show the potential of adding primal cuts to ISUD, and conclusions are drawn in Section 5.6. An extended abstract of this work was published by Rosat et al. (2014).

5.2 Literature Review and Contribution Statement

5.2.1 Notations

Before addressing the SPP, a general introduction on primal algorithms for ILP is given here. We consider a generic integer linear program

$$z^\star_{ILP} = \min_{\boldsymbol{x}\in\mathbb{R}^n} \left\{ \boldsymbol{c}^T\boldsymbol{x} \,|\, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\,,\ \boldsymbol{x} \geq 0 \text{ and } \boldsymbol{x} \text{ is integer} \right\} \tag{ILP}$$

where $\boldsymbol{A} \in \mathbb{N}^{m\times n}$, $\boldsymbol{b} \in \mathbb{N}^m$ and $\boldsymbol{c} \in \mathbb{N}^n$, with $\mathbb{N}$ the set of natural integers. The set of indices of the rows is denoted as $\mathcal{R} = \{1,\dots,m\}$. $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ are called the linear constraints and $\boldsymbol{x} \geq 0$ the nonnegativity constraints. The set of all feasible solutions of ILP is denoted by $\mathcal{F}_{\text{ILP}}$. $z^\star_{ILP}$ is called the optimal value of ILP and any feasible solution $\boldsymbol{x}^\star \in \mathcal{F}_{\text{ILP}}$ such that $\boldsymbol{c}^T\boldsymbol{x}^\star = z^\star_{ILP}$ is called an optimal solution of ILP. The linear relaxation of ILP, denoted as ILP^{LR} is the linear program obtained by relaxing the integrality constraints of ILP. Its feasible domain and optimal value are resp. denoted as $\mathcal{F}_{\text{ILP}^{LR}}$ and $z^\star_{ILP^{LR}}$. Note that in this paper, all optimization models are written in their minimization form, but the ideas and formulas also apply to maximization scenarios.

Lower-case bold symbols are used for column vectors and upper-case bold symbols denote matrices. For subsets $\mathcal{X} \subseteq \{1,\dots,m\}$ of row indices and $\mathcal{Y} \subseteq \{1,\dots,n\}$ of column indices, the submatrix of $\boldsymbol{A}$ with rows indexed by $\mathcal{X}$ and columns indexed by $\mathcal{Y}$ is denoted as $\boldsymbol{A}_{\mathcal{X}\mathcal{Y}}$. Similarly, $\boldsymbol{A}_{\mathcal{X}\boldsymbol{.}}$ is the set of rows of $\boldsymbol{A}$ indexed by $\mathcal{X}$, while $\boldsymbol{A}_{\boldsymbol{.}\mathcal{Y}}$ is the set of columns of $\boldsymbol{A}$ indexed by $\mathcal{Y}$, and for any vector $\boldsymbol{v} \in \mathbb{R}^n$, $\boldsymbol{v}_\mathcal{Y}$ is the subvector of all v_y, $y \in \mathcal{Y}$. The vector of all zeros (resp. ones) with dimension dictated by the context is denoted by $\boldsymbol{0}$ (resp. $\boldsymbol{e}$), and $\boldsymbol{A}^T$ is the transpose of $\boldsymbol{A}$. Finally, the linear span of all columns of any matrix $\boldsymbol{M}$, also called image of $\boldsymbol{M}$, is denoted as $Span\,(M)$.

5.2.2 Primal Algorithms

As noted by Letchford & Lodi (2002), algorithms for integer linear programming can be divided into three classes: *dual fractional*, *dual integral*, and *primal* methods. *Dual fractional* algorithms maintain optimality and linear-constraint feasibility at every iteration, and they

stop when integrality is reached. They are typically standard cutting plane procedures such as the algorithm of Gomory (1958). The classical branch-and-bound scheme is also based on a dual-fractional approach, in particular for the determination of lower bounds. *Dual integral* methods maintain integrality and optimality, and they terminate once the (primal) linear constraints are satisfied. Letchford and Lodi give the sole example of another algorithm of Gomory (1963). Finally, *primal algorithms* maintain feasibility (including integrality) throughout the process and stop when optimality is reached. These are in fact descent algorithms for which the improving sequence $(\boldsymbol{x}_k)_{k=1\ldots K}$ satisfies the conditions

- C1 $\boldsymbol{x}^k \in \mathcal{F}_{\text{ILP}}$;
- C2 $\boldsymbol{x}^K$ is optimal;
- C3 $\boldsymbol{c}^T\boldsymbol{x}^{k+1} < \boldsymbol{c}^T\boldsymbol{x}^k$.

Primal methods – sometimes classified as *augmenting algorithms* – were first introduced simultaneously by Ben-Israel & Charnes (1962) and Young (1965) and improved by Young (1968) and Glover (1968). In Young's method (Young, 1965, 1968), at iteration k, a simplex pivot is considered: if it leads to an integer solution, it is performed; otherwise, cuts are generated and added to the problem, thereby changing the underlying structure of the constraints. Young also developed the concept of a *augmenting* (or *improving*) vector at $\boldsymbol{x}^k$, i.e., a vector $\boldsymbol{z} \in \mathbb{R}^n$ such that $\boldsymbol{x}^k + \boldsymbol{z}$ is integer, feasible, and of lower cost than $\boldsymbol{x}^k$. From this notion comes the *integral augmentation problem* (**iAUG**) that involves finding such a direction if it exists or asserting that $\boldsymbol{x}^k$ is optimal.

iAUG	Find an improving vector $\boldsymbol{z} \in \mathbb{N}^n$ such that $(\boldsymbol{x}^k + \boldsymbol{z}) \in \mathcal{F}_{\text{ILP}}$ and $\boldsymbol{c}^T\boldsymbol{z} < 0$ or assert that $\boldsymbol{x}^k$ is optimal for ILP.

Remark. Traditionally, papers on constraint aggregation and integral simplex algorithms deal with minimization problems, whereas authors usually present generic primal algorithms for maximization problems. We therefore draw the reader's attention to the following: to retain the usual classification, we call the improving direction problem **iAUG**, although it supplies a decreasing direction. In the same way, an improvement (general term) is, in our case, a decrease. For the same reasons, we still call it an augmentation.

For the sake of readability, in this paper, we will differentiate **iAUG** (above) from the fractional augmentation **fAUG**. The latter is the relaxation of the former in which $\boldsymbol{z}$ may be fractional and $\boldsymbol{x}^k + \boldsymbol{z}$ needs only be a solution of the linear relaxation ILP^{LR}.

In the end of the 1990s, there has been a renewed interest in primal integer algorithms, inspired by Robert Weismantel as mentioned by Letchford & Lodi (2003a). Many recent

works specifically concern 0/1-LP (integer programming problems for which the variables can only take values 0 or 1). However, only a few papers have addressed the practical solution of **iAUG**, most of them considering it as an oracle. As a matter of fact, most of the rare computational work since 2000 on primal algorithms concerns the *primal separation problem*, defined as

P-SEP	Given a feasible solution $\boldsymbol{x}^k \in \mathcal{F}_{\text{ILP}}$ and an infeasible point $\boldsymbol{x}^\star$, find a hyperplane that separates $\boldsymbol{x}^\star$ from $\mathcal{F}_{\text{ILP}}$ and that is tight at $\boldsymbol{x}^k$ or assert that none exists.

In this case, $\boldsymbol{x}^\star$ is typically a vertex of the feasible domain of the linear relaxation $\mathcal{F}_{\text{ILP}^{LR}}$ obtained by performing one or several simplex pivots from $\boldsymbol{x}^k$. An example of primal separation hyperplanes is given in Figure 5.1.

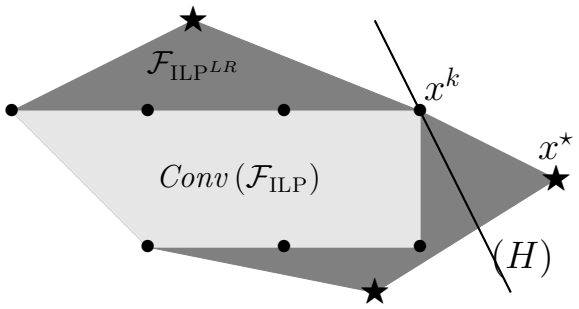


Figure 5.1 Example of primal separation. (H) is a primal cut separating $\boldsymbol{x}^\star$ from $Conv\,(\mathcal{F}_{\text{ILP}})$, and tight at $\boldsymbol{x}^k$. The dark area represents the feasible domain of the linear relaxation $\mathcal{F}_{\text{ILP}^{LR}}$. The feasible domain of ILP is a finite set represented by bullets ($\bullet$) and its convex hull $Conv\,(\mathcal{F}_{\text{ILP}})$ is the light grey polyhedron.

Eisenbrand et al. (2003) proved that the primal separation problem is, from the theoretical point of view, as difficult as the integral optimization problem for 0/1-LP. It is therefore expected to be a "complicated" problem because 0/1-LP is $\mathcal{NP}$-hard. Letchford & Lodi (2002, 2003b) and Eisenbrand et al. (2003) adapt well-known algorithms for the standard separation problem to primal separation. To the best of our knowledge, only few papers present computational experiments using primal methods. Best examples are those of Salkin & Koncal (1973), Letchford & Lodi (2002), Haus et al. (2003), and Stallmann & Brglez (2007). All these papers present results on small to mid-size instances. Haus et al. (2003) describe a solid framework and their implementation is certainly the most complete one. Letchford & Lodi (2002) present results for an algorithm using primal cutting planes and, interestingly, they stated that degeneracy prevented them from solving larger instances.

As was already mentioned above, the ISUD algorithm was originally designed to cope with degeneracy and therefore seems a promising answer to these computational limits of primal algorithms.

For further information on primal algorithms, the reader is referred to the more extensive review of Spille & Weismantel (2005).

5.2.3 The Set Partitioning Problem

The Set Partitioning Problem (SPP) is a particular case of 0/1-LP, presented in the introduction. In a mathematical programming form, it reads as

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \left\{ \boldsymbol{c}^T\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{e}\,, \mathbf{0} \leq \boldsymbol{x} \leq \boldsymbol{e} \text{ and } \boldsymbol{x} \text{ is integer} \right\} \tag{SPP}$$

where $\boldsymbol{A} \in \{0,1\}^{m\times n}$ is a {0,1}-matrix, and $\boldsymbol{c} \in \mathbb{N}^n$ is any integer cost vector. Obviously, given the bounds ($\mathbf{0}$ and $\boldsymbol{e}$) and the integrality constraints, $\mathcal{F}_{\text{SPP}}$ only contains {0,1}-vectors. As for any integer linear program, its linear relaxation SPP^{LR} is defined by relaxing the integrality constraints and the feasible domain of this linear relaxation is denoted as $\mathcal{F}_{\text{SPP}^{LR}}$. Moreover, given a current solution $\boldsymbol{x}^0 \in \mathcal{F}_{\text{SPP}}$, the indices of its components can be partitioned into sets $\mathcal{P} = \{j | x_j^0 = 1\}$, which is the set of positive-valued variables, and $\mathcal{Z} = \{j | x_j^0 = 0\}$, which is the set of null variables. Submatrix $\boldsymbol{A}_{.\mathcal{P}}$ is referred to as the *working basis*, and so is $\mathcal{P}$ by extension. Its indices and the variables of $\boldsymbol{x}_{\mathcal{P}}$ are respectively referred to as *basic indices* and *basic variables*, and $p = |\mathcal{P}|$ denotes the cardinality of this working basis. The working basis is different from a standard simplex basis because it only contains linearly independent positive-valued variables (no degenerate variables). In the rest of this paper, the terms basis, basic, etc. refer to the working basis $\mathcal{P}$.

As proved by Trubin (1969), the SPP is *quasi-integral*, i.e., every edge of the convex hull of the feasible set $Conv\,(\mathcal{F}_{\text{SPP}})$ is also an edge of the polytope of the linear relaxation $\mathcal{F}_{\text{SPP}^{LR}}$. A consequence of this property is the existence of a decreasing sequence of integer solutions of SPP leading to an optimal solution, such that two consecutive solutions are adjacent vertices of $\mathcal{F}_{\text{SPP}^{LR}}$ (easily proven by applying the simplex algorithm over $Conv\,(\mathcal{F}_{\text{SPP}})$). When working on the SPP, the following condition `C4` can therefore be added to `C1`–`C3` to transform a primal algorithm into an *integral simplex* without preventing the procedure to reach optimality.

`C4` x^{k+1} is a neighbor of x^k in $\mathcal{F}_{\text{SPP}^{LR}}$.

These methods, first introduced in 1975 by Balas & Padberg (1975), yield a sequence of

improving all-integer solutions, obtained by only performing simplex pivots. Since this seminal paper, other integral simplex methods have been proposed (Thompson, 2002; Saxena, 2003a). Amongst contemporary work conducted parallel to ours, the most interesting one to mention is that conducted by Rönnberg and Larsonn (Rönnberg & Larsson, 2009, 2014), and Rönnberg's PhD Thesis (2012) that tackles nurse scheduling problems with an integral simplex algorithm applied within a column-generation framework. Finally, note that the SPP is by nature highly degenerate. Hence it seems relevant to apply that kind of "anti-degeneracy" techniques in this case.

5.2.4 Contribution Statement

With the concepts introduced in Section 5.2, we can describe the contributions of Sections 5.3 and 5.4 more clearly. In Section 5.3, we present the ISUD algorithm in a primal way, and relate it to the augmentation problems **iAUG** and **fAUG**. We formulate **fAUG** as a linear program, and **iAUG** as a nonlinear one of which **fAUG** is the linear relaxation. Each of these problems is decomposed into two subproblems. We reduce the number of constraints of both decomposed subproblems, and give a geometrical interpretation of these row-reduced problems. We also provide a simple characterization of integer directions. Section 5.4 addresses the improvement of the linear relaxation of **iAUG** with cutting planes. We demonstrate that every valid inequality for **iAUG** can be obtained as the linear transformation of a primal cut of SPP. We prove that such a primal cut always exists and that the characterization of integer directions given in Section 5.3 remains correct after the addition of cutting planes. Finally, we introduce two new **P-SEP** procedures for primal clique and odd-cycle cuts, and show that the search space for the cuts can be reduced to a small number of variables without changing the outcome of **P-SEP**.

5.3 The Integral Simplex Using Decomposition (ISUD)

This section aims to present the Integral Simplex Using Decomposition (ISUD) of Zaghrouti et al. (2014) from a purely primal point of view, so as to make the link with primal algorithms straightforward, and simplify some proofs as well as the geometrical interpretation of the process. Section 5.3.1 concentrates on **fAUG**, while Section 5.3.2 also considers integrality and tackles **iAUG**.

Hereinafter, suppose a decreasing sequence of solutions of SPP ending at $\boldsymbol{x}^k$ is known, and **iAUG** must now be solved. For the sake of readability, we always denote the current (binary) solution as $\boldsymbol{x}^0$. We want to determine a direction $\boldsymbol{d} \in \mathbb{R}^n$ and a step $r > 0$ such

that $\boldsymbol{x}^1 = \boldsymbol{x}^0 + r\boldsymbol{d} \in \mathcal{F}_{\text{SPP}}$ and of lower cost than $\boldsymbol{x}^0$ or to assert that $\boldsymbol{x}^0$ is optimal. The set of positive-valued variables is denoted as $\mathcal{P}^0 = \{j | x_j^0 = 1\}$, and that of null variables as $\mathcal{Z}^0 = \{j | x_j^0 = 0\}$. For the sake of readability, $\mathcal{P}$, $\mathcal{Z}$, $\boldsymbol{d}$, and r (and later other objects) will not be indexed on the index of the current solution (k or 0) although they depend on $\boldsymbol{x}^0$ (or $\boldsymbol{x}^k$).

5.3.1 Fractional Augmentation fAUG and Phase Decomposition

In Section 5.3.1, the sole problem of a fractional augmentation, **fAUG**, is considered. However, for the sake of clarity, $\boldsymbol{x}^0 \in \mathcal{F}_{\text{SPP}}$ is still supposed to be integer. This section extends the work done by Omer et al. (2015), and Rosat et al. (2015a) and details it in the case of the SPP.

Generic Fractional Augmentation

To practically address **fAUG**, it must be formulated in such a way that it can algorithmically be solved. It consists in finding a direction $\boldsymbol{d} \in \mathbb{R}^n$ such that it is *feasible* ($\exists \rho > 0 | \boldsymbol{x}^k + \rho \boldsymbol{d} \in \mathcal{F}_{\text{SPP}^{LR}}$) and *augmenting* ($\boldsymbol{c}^T \boldsymbol{d} < 0$). The set of all feasible directions at $\boldsymbol{x}^0$ is the cone

$$\Gamma = \{\boldsymbol{d} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{d} = 0\,,\, \boldsymbol{d}_{\mathcal{P}} \leq \boldsymbol{0}\,,\, \boldsymbol{d}_{\mathcal{Z}} \geq \boldsymbol{0}\}\,, \tag{5.1}$$

from which we will only consider a section,

$$\Delta = \Gamma \cap \left\{\boldsymbol{d} \in \mathbb{R}^n \mid \boldsymbol{e}^T \boldsymbol{d}_{\mathcal{Z}} = 1\right\}. \tag{5.2}$$

The linear constraint $\boldsymbol{e}^T \boldsymbol{d}_{\mathcal{Z}} = 1$ is called the normalization constraint. A geometric interpretation of Γ and Δ is given in Figure 5.2. Note that, since $\boldsymbol{A}$ is nonnegative and because of the sign constraints, any nonzero feasible direction $\boldsymbol{d} \in \Gamma$ has nonzero terms in both $\boldsymbol{d}_{\mathcal{P}}$ and $\boldsymbol{d}_{\mathcal{Z}}$. Hence, the normalization constraint defines a proper section of the cone and Δ is a (bounded) polytope.

It is easy to see that at least one feasible direction is augmenting if and only if the program

$$z^\star_{\text{MIMA}} = \min_{d \in \mathbb{R}^n} \left\{\boldsymbol{c}^T \boldsymbol{d} \,|\boldsymbol{d} \in \Delta\right\} \tag{MIMA}$$

satisfies $z^\star_{\text{MIMA}} < 0$. On the one hand, any optimal solution of MIMA yields a solution to **fAUG**; on the other hand, if $z^\star_{\text{MIMA}}$ is nonnegative, no feasible augmenting direction exists and $\boldsymbol{x}^0$ is optimal for SPP. Finally, since Δ is a polytope, MIMA is a bounded linear program. The name MIMA stands for *Maximum Incoming Mean Augmentation*: The nor-

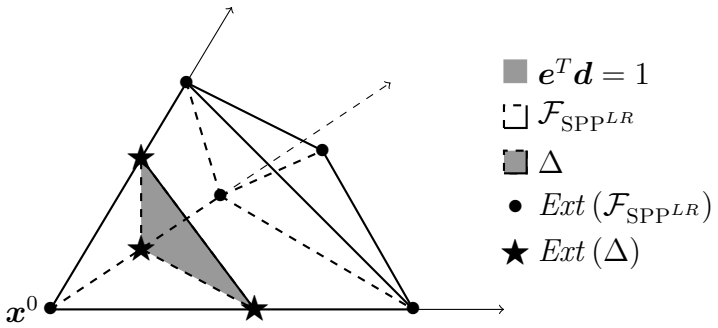


Figure 5.2 Geometric description of Δ and Γ. The cone Γ of all feasible directions at $\boldsymbol{x}^0$ is defined by the three arrowed lines. The grey area represents Δ, the set of normalized feasible directions.

malization constraint only concerns incoming variables, so the objective is a mean over a reduced subset of the variables. The choice of this normalization constraint follows that of the original algorithm of Zaghrouti et al. (2014).

Remark. The specific augmentation problem MIMA is close to the classical *maximum mean augmentation* problem (which is itself an extension of the *minimum mean cycle* (Goldberg & Tarjan, 1989) to more general linear programs; see Spille & Weismantel (2005)). They only differ in the normalization constraint: that of MIMA concerns only the nonbasic variables, while that of MMA is $-\boldsymbol{e}^T\boldsymbol{d}_{\mathcal{P}} + \boldsymbol{e}^T\boldsymbol{d}_{\mathcal{Z}} = 1$. The study of the influence of the choice of the normalization weights on the behavior of the algorithm is discussed by Rosat et al. (2015a). Hence, no further attention will be given to this matter here.

Once an optimal solution $\boldsymbol{d}^\star$ to MIMA has been found, the idea of an augmentation algorithm is to follow that direction as far as possible while remaining feasible. The *maximal feasible step* alongside direction $\boldsymbol{d}$ is thus defined as $r(\boldsymbol{d}) = \max\{\rho > 0 \,|\, \boldsymbol{x}^0 + \rho\boldsymbol{d} \in \mathcal{F}_{\mathrm{SPP}^{LR}}\}$. From $\boldsymbol{x}^0$, fractional augmentation can therefore be performed as $\boldsymbol{x}^1 = \boldsymbol{x}^0 + r(\boldsymbol{d})\boldsymbol{d}$.

Incompatibility Degree of the Nonbasic Variables

To decompose MIMA into smaller problems, both in terms of number of variables and constraints, the notion of incompatibility degree introduced by Elhallaoui et al. (2010) must be presented here. Given a column $\boldsymbol{A}_{\cdot j}$ of the constraint matrix, a row $r \in \mathcal{R}$ is said to be *covered* by that column if $A_{rj} = 1$. For each column $\boldsymbol{A}_{\cdot j}$ of $\boldsymbol{A}$, let $\mathcal{R}_j = \{r \in \mathcal{R} \,|\, A_{rj} = 1\}$ be the set of rows covered by that column. By definition, the sets of the rows covered by the columns of $\mathcal{P}$, i.e., $\{\mathcal{R}_l\}_{l\in\mathcal{P}}$, form a partition of $\mathcal{R}$ (see Figure 5.3). Assume that a total

order $\preceq$ is known over the indices of the rows $\mathcal{R}$. For any j, that order is extended to $\mathcal{R}_j$, and the elements of $\mathcal{R}_j$ are written according to $\preceq$ as

$$\mathcal{R}_j \,:\, r_1^j \preceq r_2^j \preceq \ldots \preceq r_{|\mathcal{R}_j|}^j \,.$$

Definition 6 (Incompatibility degree)**.** The *incompatibility degree* of a column $\boldsymbol{A}_{\cdot j}$, $j \in \{1, \ldots, n\}$, with respect to the working basis $\mathcal{P}$ is computed as

$$\iota_j^{\mathcal{P}} = \sum_{l \in \mathcal{P}} \sum_{t=1}^{|\mathcal{R}_l|-1} \kappa_{lj}^t \tag{5.3}$$

where $\kappa_{lj}^t = 1$ if $\boldsymbol{A}_{\cdot j}$ covers r_t^j or r_{t+1}^j but not both, 2 if $\boldsymbol{A}_{\cdot j}$ covers r_t^j and r_{t+1}^j but not consecutively, 0 otherwise. Here, r_t^j and r_{t+1}^j denote two rows covered by column l that are performed consecutively within that column. An example is given in Figure 5.3.

$$\boldsymbol{A} = \begin{array}{c|cc|ccccc} & \multicolumn{2}{c|}{\mathcal{P}} & \multicolumn{5}{c}{\mathcal{Z}} \\ & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \iota_j^{\mathcal{P}} & 0 & 0 & 1 & 1 & 2 & 1 & 2 \end{array} \qquad \hat{\boldsymbol{A}} = \begin{array}{c|cc|ccccc} & \multicolumn{2}{c|}{\mathcal{P}} & \multicolumn{5}{c}{\mathcal{Z}} \\ & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ \hline \iota_j^{\mathcal{P}} & 0 & 0 & 1 & 2 & 3 & 2 & 0 \end{array}$$

Figure 5.3 Incompatibility degree with respect to $\mathcal{P}$ on a 5-rows example for two different ordering of the rows of the same matrix, namely $(1, 2, 3, 4, 5)$ in matrix $\boldsymbol{A}$, and $(3, 4, 2, 5, 1)$ in matrix $\hat{\boldsymbol{A}}$.

Remark. Any ordering of the rows can be used with these definitions. In scheduling applications, the rows correspond to tasks to carry out. It is therefore intuitive to order them by starting time and an incompatibility will correspond to the breaking of a succession of tasks that are performed consecutively in the current solution $\boldsymbol{x}^0$.

For the sake of clarity, we will always consider that the nonzero entries of a column of the current solution are consecutive within $\mathcal{R}$, i.e., given any pair of different indices $i, j \in \mathcal{P}$, either $\forall r \in \mathcal{R}_i, \forall r' \in \mathcal{R}_j, r_i \preceq r'_j$, or $\forall r \in \mathcal{R}_i, \forall r' \in \mathcal{R}_j, r'_j \preceq r_i$. Note that this is not the case of the left part of Figure 5.3. With $\iota_j^{\mathcal{P}}$ as defined in Formula (5.3), $\boldsymbol{A}_{\cdot j}$ is said to be $\iota_j^{\mathcal{P}}$*-incompatible*. 0-incompatible columns are called *compatible* and the others are called *incompatible*. These notions extend to the index of the column and to the corresponding variable.

Observation 1. All columns from the working basis are compatible;

Observation 2. An incompatible column is one that breaks the partition of the rows $\{\mathcal{R}_l\}_{l\in\mathcal{P}}$.

Given a solution $\boldsymbol{x}^0$, we define the following subsets of nonbasic variables $\mathcal{Z}$ as

$$\mathcal{C} = \left\{ j \in \mathcal{Z} \,|\, \iota_j^{\mathcal{P}} = 0 \right\} \text{ and } \mathcal{I}^{\iota} = \left\{ j \in \mathcal{Z} \,|\, 1 \leq \iota_j^{\mathcal{P}} \leq \iota \right\}, \forall 1 \leq \iota \leq k_{max} \tag{5.4}$$

Then, $\mathcal{C}$ is called the *compatible* set and the corresponding indices and variables are called *compatible*. Thus, $\mathcal{I}^{\iota}$ is called the *at most ι-incompatible* set and the corresponding indices and variables are said to be *at most ι-incompatible*. By definition, $\mathcal{I}^1 \subseteq \mathcal{I}^2 \subseteq \ldots \subseteq \mathcal{I}^{\iota_{max}} = \mathcal{I}$. $\mathcal{I}$ is called the incompatible set, and its elements and the corresponding variables are called incompatible. Finally, $(\mathcal{P}, \mathcal{C}, \mathcal{I})$ form a partition of $\{1, \ldots, n\}$.

Lemma 18. *Given $j \in \mathcal{Z}$, the following statements are equivalent:*

(i) $\boldsymbol{A}_{\cdot j}$ *is compatible;*

(ii) $\exists \mathcal{P}_j \subseteq \mathcal{P} \,|\, \boldsymbol{A}_{\cdot j} = \sum_{l\in\mathcal{P}_j} \boldsymbol{A}_{\cdot l}$*;*

(iii) $\boldsymbol{A}_{\cdot j} \in \mathit{Span}\,(\boldsymbol{A}_{\cdot\mathcal{P}})$.

Proof. First, $(i) \Leftrightarrow (ii)$ is a straightforward consequence of Observation 2. Second, $\boldsymbol{A}$ is a {0,1}-matrix, and $\boldsymbol{A}_{\cdot\mathcal{P}}$ forms a partition of its rows, therefore $(ii) \Leftrightarrow (iii)$. □

Lemma 18 does not apply to the left part of the example given in Figure 5.3 because the rows are not ordered properly in that case. The notion of compatible/incompatible column is extended to all vectors of $\mathbb{R}^m$, namely $\boldsymbol{w} \in \mathbb{R}^m$ is compatible if and only if $\boldsymbol{w} \in \mathit{Span}\,(\boldsymbol{A}_{\cdot\mathcal{P}})$. In the theoretical part of this paper, we will consider the partition of $\{1, \ldots, n\}$ into $(\mathcal{P}, \mathcal{C}, \mathcal{I})$. However it is algorithmically efficient to look first for an augmenting direction over $\mathcal{C}$, and then, successively $\mathcal{I}^1$, $\mathcal{I}^2$, ..., until $\mathcal{I}$.

The IPS Decomposition

As previously mentioned, the set of indices of the variables $\{1, \ldots, n\}$ is partitioned as $(\mathcal{P}, \mathcal{C}, \mathcal{I})$. With $\bar{\mathcal{P}} = \{1, \ldots, m\} \setminus \mathcal{P}$ and reordering the rows and columns of $\boldsymbol{A}$ so that $\mathcal{R}$ is partitioned into $(\mathcal{P}, \bar{\mathcal{P}})$, we can write

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I}_p & \boldsymbol{A}_{\mathcal{P}\mathcal{C}} & \boldsymbol{A}_{\mathcal{P}\mathcal{I}} \\ \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}} & \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{C}} & \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{I}} \end{bmatrix} \tag{5.5}$$

where $\boldsymbol{I}_p$ is the $p \times p$ identity matrix. Given $\boldsymbol{d} \in \Delta$, from the constraints $\boldsymbol{A}\boldsymbol{d} = \boldsymbol{0}$, one can easily see that the aggregation of all columns corresponding to increasing variables (for

which $d_j \geqslant 0$) $\boldsymbol{w} = \boldsymbol{A}_{\cdot\mathcal{Z}}\boldsymbol{d}_{\mathcal{Z}}$ is compatible. As in a reduced-gradient algorithm, we are in fact looking for an aggregate column $\boldsymbol{w}$ that can enter the working basis and take a positive value by lowering only some variables of $\mathcal{P}$. We introduce the following problems, respectively called *Restricted*-Mima and *Complementary*-Mima):

$$z^\star_{\text{R-Mima}} = \min_{\boldsymbol{d}\in\mathbb{R}^{p+|\mathcal{C}|}} \left\{ \boldsymbol{c}_\mathcal{P}^T\boldsymbol{d}_\mathcal{P} + \boldsymbol{c}_\mathcal{C}^T\boldsymbol{d}_\mathcal{C} \,|\, \boldsymbol{A}_{\cdot\mathcal{P}}\boldsymbol{d}_\mathcal{P} + \boldsymbol{A}_{\cdot\mathcal{C}}\boldsymbol{d}_\mathcal{C} = 0\,,\ \boldsymbol{e}_\mathcal{C}^T\boldsymbol{d}_\mathcal{C} = 1\,,\ \boldsymbol{d}_\mathcal{C} \geq 0 \right\} \quad \text{(R-Mima)}$$

$$z^\star_{\text{C-Mima}} = \min_{\boldsymbol{d}\in\mathbb{R}^{p+|\mathcal{I}|}} \left\{ \boldsymbol{c}_\mathcal{P}^T\boldsymbol{d}_\mathcal{P} + \boldsymbol{c}_\mathcal{I}^T\boldsymbol{d}_\mathcal{I} \,|\, \boldsymbol{A}_{\cdot\mathcal{P}}\boldsymbol{d}_\mathcal{P} + \boldsymbol{A}_{\cdot\mathcal{I}}\boldsymbol{d}_\mathcal{I} = 0\,,\ \boldsymbol{e}_\mathcal{I}^T\boldsymbol{d}_\mathcal{I} = 1\,,\ \boldsymbol{d}_\mathcal{I} \geq 0 \right\} \quad \text{(C-Mima)}$$

Theorem 19. $z^\star_{\text{Mima}} = \min\left\{z^\star_{\text{R-Mima}}, z^\star_{\text{C-Mima}}\right\}$.

Proof. Let $\boldsymbol{d} = (\boldsymbol{d}_\mathcal{P}, \boldsymbol{d}_\mathcal{C}, \boldsymbol{d}_\mathcal{I})$ be an optimal solution of Mima. If $\boldsymbol{d}_\mathcal{C} = \boldsymbol{0}$ or $\boldsymbol{d}_\mathcal{I} = \boldsymbol{0}$, $\boldsymbol{d}$ is respectively a solution of C-Mima or R-Mima and the result clearly holds.

Suppose now that $\boldsymbol{d}_\mathcal{C} \neq \boldsymbol{0}$ and $\boldsymbol{d}_\mathcal{I} \neq \boldsymbol{0}$. We first prove that $\boldsymbol{d}$ can be written as a convex combination of $\boldsymbol{u}'$, a solution of R-Mima, and $\boldsymbol{v}'$, a solution of C-Mima. By Lemma 18-(ii), the surrogate column $\boldsymbol{A}_{\cdot\mathcal{C}}\boldsymbol{d}_\mathcal{C}$ can be written as a linear combination of columns of $\mathcal{P}$, $\boldsymbol{A}_{\cdot\mathcal{C}}\boldsymbol{d}_\mathcal{C} = -\boldsymbol{A}_{\cdot\mathcal{P}}\boldsymbol{u}'_\mathcal{C}$, $\boldsymbol{u}'_\mathcal{P} \leq \boldsymbol{0}$. Let $\boldsymbol{u} = (\boldsymbol{u}'_\mathcal{P}, \boldsymbol{d}_\mathcal{C}, \boldsymbol{0})$, and $\boldsymbol{v} = \boldsymbol{d} - \boldsymbol{u}$. Let $\alpha_{\boldsymbol{u}} = \|\boldsymbol{d}_\mathcal{C}\|_1 = \sum_{j\in\mathcal{C}} d_j$ and $\alpha_{\boldsymbol{v}} = \|\boldsymbol{d}_\mathcal{I}\|_1 = \sum_{j\in\mathcal{I}} d_j$. Moreover, let $\boldsymbol{u}' = \boldsymbol{u}/\alpha_{\boldsymbol{u}}$ and $\boldsymbol{v}' = \boldsymbol{v}/\alpha_{\boldsymbol{v}}$ be the corresponding normalized directions. Thus, $0 < \alpha_{\boldsymbol{u}}, \alpha_{\boldsymbol{v}}$ and, since $\boldsymbol{d}$ is a solution of Mima, the normalization constraint yields $\alpha_{\boldsymbol{u}} + \alpha_{\boldsymbol{v}} = 1$ because $\boldsymbol{d} \in \mathcal{F}_{\text{Mima}}$. Therefore, $\boldsymbol{d} = \alpha_{\boldsymbol{u}}\boldsymbol{u}' + \alpha_{\boldsymbol{v}}\boldsymbol{v}'$ is a convex combination of $\boldsymbol{u}'$ a solution of R-Mima and $\boldsymbol{v}'$ a solution of C-Mima.

Looking at the objective function, the convex combination reads $\boldsymbol{c}^T\boldsymbol{d} = \alpha_{\boldsymbol{u}}(\boldsymbol{c}^T\boldsymbol{u}') + \alpha_{\boldsymbol{v}}(\boldsymbol{c}^T\boldsymbol{v}')$. Either $\boldsymbol{c}^T\boldsymbol{d} \geq \boldsymbol{c}^T\boldsymbol{u}'$ or $\boldsymbol{c}^T\boldsymbol{d} \geq \boldsymbol{c}^T\boldsymbol{v}'$. However, since every solution of R-Mima and C-Mima is also a solution of Mima and $\boldsymbol{d}$ is optimal for Mima, $\boldsymbol{c}^T\boldsymbol{d} \leq \boldsymbol{c}^T\boldsymbol{u}'$ and $\boldsymbol{c}^T\boldsymbol{d} \leq \boldsymbol{c}^T\boldsymbol{v}'$. One of the two inequalities is thus an equality and the second follows from $\boldsymbol{c}^T\boldsymbol{d} = \alpha_{\boldsymbol{u}}(\boldsymbol{c}^T\boldsymbol{u}') + \alpha_{\boldsymbol{v}}(\boldsymbol{c}^T\boldsymbol{v}')$. Therefore, $\boldsymbol{c}^T\boldsymbol{d} = \boldsymbol{c}^T\boldsymbol{u}' = \boldsymbol{c}^T\boldsymbol{v}'$, and since $\boldsymbol{d}$ is an optimal solution of Mima, $z^\star_{\text{Mima}} = z^\star_{\text{R-Mima}} = z^\star_{\text{C-Mima}}$. □

Theorem 19 extends the results of Elhallaoui et al. (2011) and Zaghrouti et al. (2014), and it also justifies their procedures in a trivial way. This purely primal interpretation of their less-intuitive dual approach allows us to state a precise decomposition of Mima into R-Mima and C-Mima. As a consequence of Theorem 19, we will consider the pair of problems R-Mima and C-Mima instead of the more complicated Mima, and we will

solve them sequentially. In particular, C-MIMA will not be solved if $z^\star_{\text{R-MIMA}} < 0$ because an improving direction is already known. In the next sections, we discuss how to reduce the number of rows in R-MIMA and C-MIMA to ease their solution. Recall that, for the moment, $\boldsymbol{x}^1 = \boldsymbol{x}^0 + r(\boldsymbol{d})\boldsymbol{d}$ may be fractional.

Row-reduction of R-MIMA

The practical solution of the restriction of **fAUG** to the compatible variables only is based on the following proposition. Recall that for all $i \in \mathcal{C}$, $\mathcal{P}_i \subseteq \mathcal{P}$ is defined as in Lemma 18-(ii), i.e., it is the unique subset of $\mathcal{P}$ such that $\boldsymbol{A}_{\cdot i} = \sum_{j\in\mathcal{P}_i} \boldsymbol{A}_{\cdot j}$.

Proposition 20. *Given $j \in \mathcal{C}$, the* minimal direction *associated with j is the vector $\boldsymbol{\delta}^j \in \mathbb{R}^{p+|\mathcal{C}|}$ defined as*

$$\forall i \in \mathcal{P}\cup\mathcal{C}\,,\ \delta^j_i = \begin{cases} -1 & \text{if } i \in \mathcal{P}_j, \\ 1 & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \quad . \tag{5.6}$$

For any $j \in \mathcal{C}$, $\boldsymbol{\delta}^j$ is feasible and the set of all extreme points of $\mathcal{F}_{\text{R-MIMA}}$ is exactly the set of the minimal directions associated with variables of $\mathcal{C}$, i.e., $\left\{\boldsymbol{\delta}^j | j \in \mathcal{C}\right\}$. Moreover, the maximal feasible step alongside direction $\boldsymbol{\delta}^j$ is $r(\boldsymbol{\delta}^j) = 1$.

Proof. First, let $j \in \mathcal{C}$. By definition of $\boldsymbol{\delta}^j$ and $\mathcal{F}_{\text{R-MIMA}}$, $\boldsymbol{\delta}^j$ is obviously feasible for R-MIMA.

Second, let $\boldsymbol{d}^\star = (\boldsymbol{d}^\star_{\mathcal{P}}, \boldsymbol{d}^\star_{\mathcal{C}}) \in \mathcal{F}_{\text{R-MIMA}}$ be a feasible solution of R-MIMA. We will show that $\boldsymbol{d}$ is a convex combination of one or several minimal solutions. Let $\boldsymbol{u} = \sum_{j\in\mathcal{C}} d^\star_j \boldsymbol{\delta}^j$. By construction, $\boldsymbol{u}_{\mathcal{C}} = \boldsymbol{d}^\star_{\mathcal{C}}$. Because all $\boldsymbol{\delta}^j$ are in $\mathcal{F}_{\text{R-MIMA}}$, then the linearity constraints apply to their combination $\boldsymbol{u}$. Hence, $\boldsymbol{A}_{\cdot\mathcal{P}}\boldsymbol{u}_{\mathcal{P}} = -\boldsymbol{A}_{\cdot\mathcal{C}}\boldsymbol{u}_{\mathcal{C}} = -\boldsymbol{A}_{\cdot\mathcal{C}}\boldsymbol{d}^\star_{\mathcal{C}} = \boldsymbol{A}_{\cdot\mathcal{P}}\boldsymbol{d}^\star_{\mathcal{P}}$. The columns of the working basis $\boldsymbol{A}_{\cdot\mathcal{P}}$ are linearly independent, therefore, $\boldsymbol{u}_{\mathcal{P}} = \boldsymbol{d}^\star_{\mathcal{P}}$ and $\boldsymbol{d}^\star = \boldsymbol{u}$.

Moreover, $\boldsymbol{d} \in \mathcal{F}_{\text{R-MIMA}}$ satisfies the normalization constraint $\boldsymbol{e}^T\boldsymbol{d}^\star_{\mathcal{C}} = \sum_{i\in\mathcal{C}} d^\star_i = 1$. Thus, $\boldsymbol{d}^\star = \boldsymbol{u} = \sum_{j\in\mathcal{C}} d^\star_j\boldsymbol{\delta}^j$ is a convex combination of minimal directions. Finally, given the current solution $\boldsymbol{x}^0$, for any $j \in \mathcal{C}$ and $\rho > 0$,

$$\forall i \in \mathcal{P}\cup\mathcal{C}\,,\ \left[\boldsymbol{x}^0 + \rho\boldsymbol{\delta}^j\right]_i = \begin{cases} 1-\rho & \text{if } i \in \mathcal{P}_j, \\ \rho & \text{if } i = j, \\ x^0_i & \text{otherwise.} \end{cases}$$

Therefore, considering the bounds $\boldsymbol{0} \le (\boldsymbol{x}^0 + \rho\boldsymbol{d}) \le \boldsymbol{e}$, the maximum possible value for ρ is $r(\boldsymbol{d}^0) = 1$. □

Corollary 21. *There exists $j \in \mathcal{P}$ such that $\boldsymbol{\delta}^j$ is an optimal solution of* R-MIMA.

As a consequence of Proposition 20 and Corollary 21, solving R-MIMA and following the optimal direction until a bound is reached is equivalent to pivoting the corresponding compatible variable into the working basis. These pivots are guaranteed to be nondegenerate, as proven in the following proposition.

Proposition 22. *Compatible variables are exactly those that yield nondegenerate pivots when inserted in the working basis.*

Proof. For SPP, when inserted in the working basis, a nonbasic variable yields a nondegenerate pivot iff it can be written as a nonnegative linear combination of the variables in $\mathcal{P}$ ($\boldsymbol{x}^0_{\mathcal{P}} = \mathbf{0}$ and $\mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}$). By Lemma 18-(*iii*), compatible columns are exactly those that can be written as a linear combination of the columns of $\boldsymbol{A}_{\cdot\mathcal{P}}$. Therefore, if a column yields a nondegenerate pivot, it must be compatible. As proven in Proposition 20 pivoting a compatible variable x_j in the working basis can be interpreted as following direction $\boldsymbol{\delta}^j$. Since the corresponding maximal step is positive ($r(\boldsymbol{\delta}^j) = 1$), the pivot is nondegenerate and compatible variables all yield nondegenerate pivots. □

Proposition 23. R-MIMA *is equivalent to the minimization program*

$$z^\star_{\text{R-MIMA}} = z^\star_{\text{RP}} = \min_{j \in \mathcal{C}} \{\bar{c}_j\}\,, \tag{RP}$$

called reduced problem*, where* $\bar{c} = \boldsymbol{c} - \boldsymbol{c}^T_{\mathcal{P}}\boldsymbol{A}_{\mathcal{P}}$*. is the reduced costs vector. Moreover, given an optimal solution* $j \in \mathcal{C}$*, the corresponding direction is* $\boldsymbol{\delta}^j$*.*

Proof. Given an index $j \in \mathcal{C}$, the cost of the corresponding minimal direction in R-MIMA is $\boldsymbol{c}^T\boldsymbol{\delta}^j = \boldsymbol{c}^T_{\mathcal{P}}\boldsymbol{\delta}^j_{\mathcal{P}} + \boldsymbol{c}^T_{\mathcal{C}}\boldsymbol{\delta}^j_{\mathcal{C}}$. With Equation (5.5), the linear constraints of R-MIMA become

$$\begin{cases} \boldsymbol{I}_p\boldsymbol{\delta}^j_{\mathcal{P}} & +\boldsymbol{A}_{\mathcal{P}\mathcal{C}}\boldsymbol{\delta}^j_{\mathcal{C}} & = 0 \\ \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{\delta}^j_{\mathcal{P}} & +\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{C}}\boldsymbol{\delta}^j_{\mathcal{C}} & = 0 \end{cases},$$

and, with Equation (5.6), the first row gives $\boldsymbol{\delta}^j_{\mathcal{P}} = -\boldsymbol{A}_{\mathcal{P}j}$, and thus, $\boldsymbol{c}^T\boldsymbol{\delta}^j = c_j - \boldsymbol{c}^T_{\mathcal{P}}\boldsymbol{A}_{\mathcal{P}j}$. Because the extreme points of $\mathcal{F}_{\text{R-MIMA}}$ are the $\boldsymbol{\delta}^j$ for $j \in \mathcal{C}$, the result holds. □

The term of row-reduction refers to the following two facts. First, the linear program becomes a simple reduced-cost determination. Second, the computation of the whole improving direction is made without any matrix multiplication since $\boldsymbol{\delta}^j_{\mathcal{P}} = -\boldsymbol{A}_{\mathcal{P}j}$. This is in the spirit of a standard simplex pivot in which a reduced-cost analysis is performed, and then a system must be solved to determine the future solution $\boldsymbol{x}^{k+1}$. As in Proposition 23, in

practice, the determination of $j \in \mathcal{C}$ is made as a reduced-cost analysis and exactly reproduces what a simplex pivot would be. Namely, the nonbasic variable of lowest reduced-cost $z^\star_{\text{R-Mima}}$ is determined, and then, if it satisfies $z^\star_{\text{R-Mima}} < 0$, a pivot is performed by inserting that variable into the working basis. However, if $z^\star_{\text{R-Mima}} \geq 0$, it becomes necessary to consider C-Mima to determine an augmenting direction. This process is the topic of the following section.

Row-reduction of C-Mima

In this section, we suppose that the compatible variables yield no improvement, or equivalently that $\mathcal{C} = \emptyset$. The first p constraints of C-Mima read $\boldsymbol{d}_\mathcal{P} = -\boldsymbol{A}_{\mathcal{P}\mathcal{I}}\boldsymbol{d}_\mathcal{I}$. As in a reduced gradient algorithm (Kallio & Porteus, 1978), the modification of the basic variables is inferred via a linear transformation of the increasing nonbasic variables $\boldsymbol{d}_\mathcal{Z}$, or in the case of C-Mima, $\boldsymbol{d}_\mathcal{I}$. Hence, we reduce C-Mima to an equivalent problem over the nonbasic variables of $\mathcal{I}$ only. For the sake of clarity, denote as $\Delta_\mathcal{I}$ the feasible domain of C-Mima, i.e., all elements of Δ that satisfy $\boldsymbol{d}_\mathcal{C} = 0$, and define $q = |\mathcal{I}|$. That domain can be defined by less linear constraints and variables than Δ as shown by Proposition 24.

Proposition 24. *With* $\bar{\Delta_\mathcal{I}} = \left\{\boldsymbol{d}_\mathcal{I} \in \mathbb{R}^q \,|\, \bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{I}}\boldsymbol{d}_\mathcal{I} = \boldsymbol{0}\,,\, \boldsymbol{e}^T\boldsymbol{d}_\mathcal{I} = 1\,,\, \boldsymbol{d}_\mathcal{I} \geq \boldsymbol{0}\right\}$, *then*

$$\Delta_\mathcal{I} = \left\{\boldsymbol{d} = \boldsymbol{T}\boldsymbol{d}_\mathcal{I} \,|\, \boldsymbol{d}_\mathcal{I} \in \bar{\Delta_\mathcal{I}}\right\} \tag{5.7}$$

where $\boldsymbol{T} = \begin{bmatrix} -\boldsymbol{A}_{\mathcal{P}\mathcal{I}} \\ \boldsymbol{I}_q \end{bmatrix}$ *and* $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{I}} = -\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{I}} + \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{I}} = \boldsymbol{A}_{\bar{\mathcal{P}}\cdot}\boldsymbol{T}$.

Proof. Let $\boldsymbol{d} \in \mathbb{R}^n$ such that $\boldsymbol{d}_\mathcal{C} = \boldsymbol{0}$. Then,

$$\begin{aligned}
\boldsymbol{d} \in \Delta_\mathcal{I} \quad &\Leftrightarrow \quad \boldsymbol{A}_{\cdot\mathcal{P}}\boldsymbol{d}_\mathcal{P} + \boldsymbol{A}_{\cdot\mathcal{I}}\boldsymbol{d}_\mathcal{I} = 0\,,\, \boldsymbol{e}^T\boldsymbol{d}_\mathcal{I} = 1\,,\, \boldsymbol{d}_\mathcal{P} \leq \boldsymbol{0}\,,\, \boldsymbol{d}_\mathcal{I} \geq \boldsymbol{0} \\
&\Leftrightarrow \quad \begin{cases}
\boldsymbol{d}_\mathcal{P} + \boldsymbol{A}_{\mathcal{P}\mathcal{I}}\boldsymbol{d}_\mathcal{I} = \boldsymbol{0} \\
\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{d}_\mathcal{P} + \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{I}}\boldsymbol{d}_\mathcal{I} = \boldsymbol{0} \\
\boldsymbol{e}^T\boldsymbol{d}_\mathcal{I} = 1 \\
\boldsymbol{d}_\mathcal{P} \leq 0 \qquad \boldsymbol{d}_\mathcal{I} \geq 0
\end{cases} \\
&\Leftrightarrow \quad \begin{cases}
\boldsymbol{d}_\mathcal{P} = -\boldsymbol{A}_{\mathcal{P}\mathcal{I}}\boldsymbol{d}_\mathcal{I} \\
(-\boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{P}}\boldsymbol{A}_{\mathcal{P}\mathcal{I}} + \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{I}})\,\boldsymbol{d}_\mathcal{I} = \boldsymbol{0} \\
\boldsymbol{e}^T\boldsymbol{d}_\mathcal{I} = 1 \\
\boldsymbol{d}_\mathcal{I} \geq 0
\end{cases} \\
&\Leftrightarrow \quad \boldsymbol{d} = \boldsymbol{T}\boldsymbol{d}_\mathcal{I} \text{ and } \boldsymbol{d}_\mathcal{I} \in \bar{\Delta_\mathcal{I}}.
\end{aligned}$$

Hence, the proposition holds. □

The cost of such a direction can be computed as $\boldsymbol{c}^T\boldsymbol{d} = \boldsymbol{c}^T\boldsymbol{T}\boldsymbol{d}_{\mathcal{I}}$. Define $\bar{\boldsymbol{c}}^T = \boldsymbol{c}^T\boldsymbol{T} = \boldsymbol{c}_{\mathcal{I}}^T - \boldsymbol{c}_{\mathcal{P}}^T\boldsymbol{A}_{\mathcal{P}\mathcal{I}}$, and C-Mima is equivalent to the following program, called the *complementary problem*:

$$z^\star_{\text{C-Mima}} = z^\star_{\text{CP}} = \min\left\{\bar{c}^T\boldsymbol{d}_{\mathcal{I}} \,|\, \boldsymbol{d}_{\mathcal{I}} \in \bar{\Delta}_{\mathcal{I}}\right\}. \tag{CP}$$

The cost vector of CP is the reduced-costs vector associated with all incompatible variables, i.e., their own cost ($\boldsymbol{c}_{\mathcal{I}}^T$) minus the marginal impact they have on the values of the variables of $\mathcal{P}$ if they enter the reduced-basis ($-\boldsymbol{c}_{\mathcal{P}}^T\boldsymbol{A}_{\mathcal{P}\mathcal{I}}$).

Remark. In Proposition 20, for $j \in \mathcal{C}$, we called $\boldsymbol{\delta}^j$ a *minimal direction*. This denomination can be extended to directions of $\bar{\Delta}_{\mathcal{I}}$ (while still applying to those of $\bar{\Delta}_{\mathcal{C}}$) as follows: $\boldsymbol{d} \in \bar{\Delta}_{\mathcal{I}}$ is a *minimal direction* iff there exists no other direction whose support is strictly contained in that of $\boldsymbol{d}$. In that sense, the set of minimal directions of $\bar{\Delta}$ is exactly $Ext\left(\bar{\Delta}_{\mathcal{C}}\right) \cup Ext\left(\bar{\Delta}_{\mathcal{I}}\right)$ (see Rosat et al. (2015a) for more details on the geometrical structure of Δ and $\bar{\Delta}$).

5.3.2 Integral Augmentation iAUG

The previous section provides a method to find a fractional augmentation. If $z^\star_{\text{RP}} < 0$ or $z^\star_{\text{CP}} < 0$, then the corresponding solution of negative reduced cost $\boldsymbol{d} \in \Delta$ yields a new solution $\boldsymbol{x}^1 = \boldsymbol{x}^0 + r(\boldsymbol{d})\boldsymbol{d} \in \mathcal{F}_{\text{SPP}^{LR}}$. However, nothing guarantees that $\boldsymbol{x}^1$ is integral. In this section, we characterize directions that lead to an integral solution $\boldsymbol{x}^1$. Such directions are called *integral directions* as opposed to *fractional directions*. The set of all integral directions within Δ is denoted as Δ^{int}. These names apply to $\boldsymbol{d}$, and its restrictions $\boldsymbol{d}_{\mathcal{Z}}$, $\boldsymbol{d}_{\mathcal{C}}$ or $\boldsymbol{d}_{\mathcal{I}}$ depending on the context. We also give some insights on the structure of the set of all integral directions, and a generic framework for our algorithm.

Over Compatible Variables

Note that for any $j \in \mathcal{C}$, the feasible solution $\boldsymbol{\delta}^j$ of RP is an integral direction. Indeed, $\boldsymbol{A}_{\cdot j} = \sum_{i\in\mathcal{P}_j} \boldsymbol{A}_{\cdot i}$ and following a step of length $r(\boldsymbol{\delta}^j) = 1$ in that direction is equivalent to let j enter the working basis $\mathcal{P}$ and let $\mathcal{P}_j$ leave it. Namely,

$$\left[\boldsymbol{x}^0 + \rho\boldsymbol{\delta}^j\right]_i = \begin{cases} 1 & \text{if } i \in \mathcal{P} \setminus \mathcal{P}_j \\ 1-\rho & \text{if } i \in \mathcal{P}_j \\ \rho & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

The corresponding maximal step is $\rho = r(\boldsymbol{\delta}^j) = 1$, so $\boldsymbol{x}^1 = \boldsymbol{x}^0 + \boldsymbol{\delta}^j \in \mathcal{F}_{\text{SPP}}$ and integrality is maintained.

Characterization of Integral Directions in $\Delta_{\mathcal{I}}$

Now, let us consider the problem for C-Mima (and CP). We base our analysis on previous results of Zaghrouti et al. (2014) that we first recall here. For any polyhedron P, denote as $Ext\,(P)$ the set of its extreme points (or vertices).

Definition 7. A set $\mathcal{S}$ of indices of the variables in $\{1, \ldots, n\}$ is *column-disjoint* if no pair of columns of $\boldsymbol{A}_{\cdot\mathcal{S}}$ has a common nonzero entry, i.e., $\forall i, j \in \mathcal{S}, i \neq j, \forall k \in \{1, \ldots, m\}\,, A_{ki}A_{kj} = 0$. This definition is extended to $\boldsymbol{A}_{\cdot\mathcal{S}}$ and $\boldsymbol{x}_{\mathcal{S}}$. Since $\boldsymbol{A}$ is binary, $\mathcal{S}$ is column-disjoint iff $\boldsymbol{A}_{\cdot\mathcal{S}}$ is a set of orthogonal columns.

Proposition 25. (Propositions 6 and 7, Zaghrouti et al. (2014)) *Given $\boldsymbol{d} \in Ext\,(\Delta)$, $\boldsymbol{d}$ is an integral direction iff the support of $\boldsymbol{d}_{\mathcal{I}}$, denoted as $S = Supp\,(\boldsymbol{d}_{\mathcal{I}})$, is column-disjoint. In this case, $r(\boldsymbol{d}) = |S|$.*

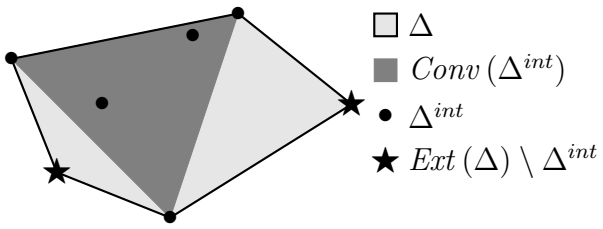


Figure 5.4 Geometrical insight on the extreme points of Δ and Δ^{int}. All extreme points of $Conv\,(\Delta^{int})$ ($\bullet$) are extreme points of Δ. Moreover, Δ^{int} is a finite set ($\mathcal{F}_{\text{SPP}}$ is finite).

Because SPP is quasi-integral, then the edges of $Conv\,(\mathcal{F}_{\text{SPP}})$ are edges of $\mathcal{F}_{\text{SPP}^{LR}}$, and $Ext\,(\Delta^{int}_{\mathcal{I}}) \subseteq Ext\,(\Delta)$. A geometrical description of this is shown on Figure 5.4. Since every linear program has at least one optimal solution that is also an extreme point of its feasible domain, and since the simplex algorithm guarantees to find such a solution, Proposition 25 has a strong practical interest. In the next section, we will show that it still remains valid when cutting planes or branching techniques are used. Hence, it is sufficient to test whether the solution of the relaxation (CP) is column-disjoint to determine if it is integral.

Algorithmic Framework

Algorithm 3 is based on successive resolutions of augmenting problems either to $\mathcal{C}$ or to $\mathcal{I}^{\iota}$ for a certain incompatibility degree ι. CP$^{\iota}$ describes the restriction of CP to the columns

Algorithm 3: Integral Simplex Using Decomposition

Input: $\boldsymbol{x}^0$, a solution of SPP; INC_MAX, maximal incompatibility degree considered.

Output: $\boldsymbol{x}^k$, a possibly better solution of SPP.

Compute $\mathcal{P}$ and $\mathcal{C}$ associated with $\boldsymbol{x}^0$; $k \leftarrow 0$; $\iota \leftarrow 1$;
while *true* **do**
  If necessary, update $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{I}^\iota$ associated with $\boldsymbol{x}^k$;
  if $z^\star_{\mathrm{RP}} < 0$ **then**
    $\boldsymbol{\delta}^i \leftarrow$ an optimal solution of RP ($\bar{c}_i < 0$, $i \in \mathcal{C}$);
    $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + \boldsymbol{\delta}^i$; $k \leftarrow k + 1$;
  else
    continue $\leftarrow$ true;
    while *continue = true* **do**
      If necessary, update $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{I}^\iota$ associated with $\boldsymbol{x}^k$, and CP^ι;
      if $z^\star_{\mathrm{CP}^\iota} < 0$ **then**
        $\boldsymbol{d}^\star_{\mathcal{I}} \leftarrow$ an optimal solution of CP^ι; $\boldsymbol{d}^\star \leftarrow \boldsymbol{T}\boldsymbol{d}^\star_{\mathcal{I}}$;
        if $\boldsymbol{d}^\star_{\mathcal{I}}$ *is column-disjoint* **then**
          $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$; $k \leftarrow k + 1$; continue $\leftarrow$ false;
        else
          if *some stopping criterion is reached* **then**
            return $\boldsymbol{x}^k$;
          else
            — Add cutting planes to CP^ι;
            — or use branching strategy;
            — or increase ι (only if $\iota < INC\_MAX$);
            — or continue $\leftarrow$ false;
      else
        if $\iota < INC\_MAX$ **then**
          $\iota \leftarrow \iota + 1$;
        else
          return $\boldsymbol{x}^k$;

of $\mathcal{I}^\iota$. INC_MAX describes the largest incompatibility degree considered during the execution. The solution returned by Algorithm 3 may not be optimal: This strongly depends on the value of INC_MAX, and on the chosen branching and cutting planes techniques. For instance, if $INC\_MAX < \iota_{max}$, there may exist augmenting integral directions involving columns of incompatibility degree greater than INC_MAX. The same holds when the branching technique is nonexhaustive.

5.4 Solving the Augmentation Problem with Cutting-Planes

In this section, we suppose that we know an optimal solution $\boldsymbol{d}^\star_{\mathcal{I}}$ of CP, but that this solution is not integral ($Supp\,(\boldsymbol{d}^\star_{\mathcal{I}})$ is not column-disjoint). An idea to tighten the known relaxation of $\bar{\Delta}^{int}_{\mathcal{I}}$ is to add cutting planes to CP. Given a polyhedron P, a valid inequality for P is an inequality satisfied by all elements of P; and given a point $\boldsymbol{x}'$, the inequality *separates* $\boldsymbol{x}'$ from P if it is valid for P but violated by $\boldsymbol{x}'$. Such an inequality is called a cut (or cutting plane). The main issue is to characterize valid inequalities for $\bar{\Delta}^{int}_{\mathcal{I}}$ that cut off the current optimal solution $\boldsymbol{d}^\star_{\mathcal{I}}$.

Notation. Denote as $\mathcal{H}^1_= = \left\{\boldsymbol{d}_{\mathcal{I}} \in \mathbb{R}^q \,|\, \boldsymbol{e}^T\boldsymbol{d}_{\mathcal{I}} = 1\right\}$ the hyperplane defined by the normalization constraint of $\bar{\Delta}_{\mathcal{I}}$. Let $\bar{\boldsymbol{\alpha}} \in \mathbb{R}^q$, $\bar{\beta} \in \mathbb{R}$, and denote by $\left(\bar{\Gamma}\right)$ the inequality

$$\left(\bar{\Gamma}\right) \,:\, \bar{\boldsymbol{\alpha}}^T\boldsymbol{d}_{\mathcal{I}} \leq \bar{\beta}. \tag{5.8}$$

The associated hyperplane is denoted as $\mathcal{H}^{\bar{\Gamma}}_= = \left\{\boldsymbol{x} \in \mathbb{R}^q | \bar{\boldsymbol{\alpha}}^T\boldsymbol{d}_{\mathcal{I}} = \bar{\beta}\right\}$ and the associated halfspace as $\mathcal{H}^{\bar{\Gamma}}_\leq = \left\{\boldsymbol{x} \in \mathbb{R}^q | \bar{\boldsymbol{\alpha}}^T\boldsymbol{d}_{\mathcal{I}} \leq \bar{\beta}\right\}$. Without loss of generality, we can assume that $\bar{\boldsymbol{\alpha}}$ is not proportional to $\boldsymbol{e}$.

Definition 8. Let $\left(\bar{\boldsymbol{\alpha}}_1, \bar{\beta}_1\right), \left(\bar{\boldsymbol{\alpha}}_2, \bar{\beta}_2\right) \in \mathbb{R}^q \times \mathbb{R}$, and $\left(\bar{\Gamma}_1\right)$ and $\left(\bar{\Gamma}_2\right)$ be the corresponding inequalities. $\left(\bar{\Gamma}_1\right)$ and $\left(\bar{\Gamma}_2\right)$ are *equivalent* for $\bar{\Delta}^{int}_{\mathcal{I}}$ if

$$\mathcal{H}^1_= \cap \mathcal{H}^{\bar{\Gamma}_1}_\leq = \mathcal{H}^1_= \cap \mathcal{H}^{\bar{\Gamma}_2}_\leq$$

Since $\bar{\Delta}_{\mathcal{I}} \subseteq \mathcal{H}^1_=$, two equivalent inequalities exactly discard the same subset of $\bar{\Delta}_{\mathcal{I}}$ and are simultaneously valid. Hence, adding the one or the other to the formulation is equivalent.

Proposition 26. *There exists* $\bar{\boldsymbol{\alpha}}' \in \mathbb{R}^q$ *such that* $\left(\bar{\Gamma}'\right) : (\bar{\boldsymbol{\alpha}}')^T\boldsymbol{d}_{\mathcal{I}} \leq 0$ *is equivalent to* $\left(\bar{\Gamma}\right)$.

Proof. Consider $\mathcal{E}_= = \mathcal{H}^1_= \cap \mathcal{H}^{\bar{\Gamma}}_=$. The two hyperplanes $\mathcal{H}^1_=$ and $\mathcal{H}^{\bar{\Gamma}}_=$ are not parallel, so $\dim(\mathcal{E}_=) = q - 2$. $\mathbf{0} \notin \mathcal{H}^1_=$, therefore $\mathbf{0} \notin \mathcal{E}_=$ and $\mathcal{H}^0_= = Span\,(\mathcal{E}_= \cup \{\mathbf{0}\})$ is a hyperplane con-

taining $\mathbf{0}$. Thus, there exists $\bar{\boldsymbol{\alpha}}' \in \mathbb{R}^q$ such that $\mathcal{H}^0_=$ is defined by $(\bar{\boldsymbol{\alpha}}')^T \boldsymbol{d}_\mathcal{I} = 0$. Furthermore, by construction, $\mathcal{H}^0_= \cap \mathcal{H}^1_= = \mathcal{E}_= = \mathcal{H}^{\bar{\Gamma}}_= \cap \mathcal{H}^1_=$. With $\bar{\boldsymbol{\alpha}}' = \pm\bar{\boldsymbol{\alpha}}^0$ (sign to be well chosen), the proposition holds. □

The geometrical interpretation of Proposition 26 is given on Figure 5.5. Hereinafter, we suppose that $\bar{\beta} = 0$. We can now characterize valid inequalities for $\bar{\Delta}^{int}_\mathcal{I}$.

Proposition 27. *Given $\boldsymbol{\alpha} \in \mathbb{R}^n$ such that $\bar{\boldsymbol{\alpha}} = \boldsymbol{T}^T\boldsymbol{\alpha}$, $\left(\bar{\Gamma}\right)$ is a valid inequality for $\bar{\Delta}^{int}_\mathcal{I}$ if and only if*

$$(\Gamma) \ : \ \boldsymbol{\alpha}^T \left(\boldsymbol{x} - \boldsymbol{x}^0\right) \leq 0 \tag{5.9}$$

is a valid inequality for $\mathcal{F}_{\text{SPP}}$.

Proof. Let $\boldsymbol{\alpha} \in \mathbb{R}^n$ such that $\bar{\boldsymbol{\alpha}} = \boldsymbol{T}^T\boldsymbol{\alpha}$. Recall Proposition 24: $\Delta_\mathcal{I} = \left\{\boldsymbol{d} = \boldsymbol{T}\boldsymbol{d}_\mathcal{I} \,|\, \boldsymbol{d}_\mathcal{I} \in \bar{\Delta}_\mathcal{I}\right\}$. This extends to $\Delta^{int}_\mathcal{I}$ and $\bar{\Delta}^{int}_\mathcal{I}$. Hence,

$$\begin{aligned}
\left(\bar{\Gamma}\right) \text{ is valid for } \bar{\Delta}^{int}_\mathcal{I} &\Leftrightarrow \forall \boldsymbol{d}_\mathcal{I} \in \bar{\Delta}^{int}_\mathcal{I}, \bar{\boldsymbol{\alpha}}^T\boldsymbol{d}_\mathcal{I} \leq 0 \\
&\Leftrightarrow \forall \boldsymbol{d}_\mathcal{I} \in \bar{\Delta}^{int}_\mathcal{I}, \boldsymbol{\alpha}^T\boldsymbol{T}\boldsymbol{d}_\mathcal{I} \leq 0 \\
&\Leftrightarrow \forall \boldsymbol{d} \in \Delta^{int}_\mathcal{I}, \boldsymbol{\alpha}^T\boldsymbol{d} \leq 0 \\
&\Leftrightarrow \forall \boldsymbol{d} \in \Delta^{int}_\mathcal{I}, \boldsymbol{\alpha}^T\left(\boldsymbol{x}^0 + r(\boldsymbol{d})\boldsymbol{d}\right) \leq \boldsymbol{\alpha}^T\boldsymbol{x}^0 \\
&\Leftrightarrow \forall \boldsymbol{x}' \in \mathcal{F}_{\text{SPP}}, \boldsymbol{\alpha}^T\left(\boldsymbol{x}' - \boldsymbol{x}^0\right) \leq 0 \\
&\Leftrightarrow (\Gamma) \text{ is valid for } SPP.
\end{aligned}$$

□

Proposition 28. *Let (Γ) and $\left(\bar{\Gamma}\right)$ be valid inequalities defined as in Proposition 27, and $\boldsymbol{d}^\star_\mathcal{I} \in \Delta_\mathcal{I}$, $\boldsymbol{d}^\star = \boldsymbol{T}\boldsymbol{d}_\mathcal{I}$, and $\boldsymbol{x}^\star = \boldsymbol{x}^0 + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$. Then,*

$$\left(\bar{\Gamma}\right) \textit{ separates } \boldsymbol{d}^\star_\mathcal{I} \textit{ from } \bar{\Delta}^{int}_\mathcal{I} \Leftrightarrow (\Gamma) \textit{ separates } \boldsymbol{x}^\star \textit{ from } \mathcal{F}_{\text{SPP}}$$

Proof. We only need to prove that $\left(\bar{\Gamma}\right)$ is violated by $\boldsymbol{d}^\star_\mathcal{I}$ if and only if (Γ) is violated by $\boldsymbol{x}^\star$. We have

$$\begin{aligned}
\bar{\boldsymbol{\alpha}}^T\boldsymbol{d}^\star > 0 &\Leftrightarrow \boldsymbol{\alpha}^T\boldsymbol{d}^\star > 0 \\
&\Leftrightarrow \boldsymbol{\alpha}^T\left(\boldsymbol{x}^0 + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star\right) > \boldsymbol{\alpha}^T\boldsymbol{x}^0 \\
&\Leftrightarrow \boldsymbol{\alpha}^T\left(\boldsymbol{x}^\star - \boldsymbol{x}^0\right) > 0
\end{aligned}$$

□

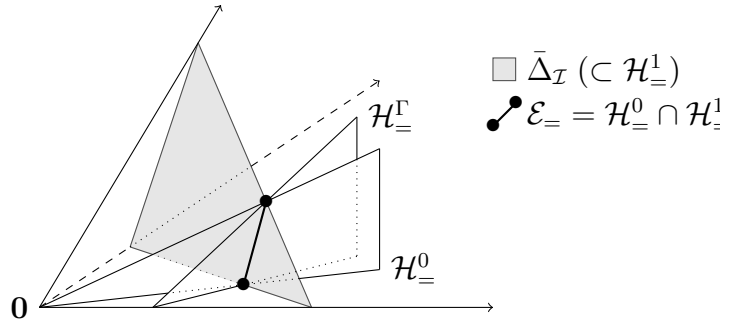


Figure 5.5 Equivalent inequalities. Both $\mathcal{H}^0_=$ and $\mathcal{H}^\Gamma_=$ cut off the same part of $\bar{\Delta}_{\mathcal{I}}$. $\mathcal{H}^0_=$ is of the form $\bar{\alpha}^T \boldsymbol{d}_{\mathcal{I}} \leq 0$ but not $\mathcal{H}^\Gamma_=$ ($\mathbf{0} \notin \mathcal{H}^\Gamma_=$).

What we have shown must now be seen the other way round to take advantage of previous work on primal separation. Assume that we know how to determine a primal cut (Γ) for SPP that separates $\boldsymbol{x}^\star$ from $\mathcal{F}_{\text{SPP}}$. Then, with $\bar{\boldsymbol{\alpha}} = \boldsymbol{T}^T\boldsymbol{\alpha}$, the associated inequality $\left(\bar{\Gamma}\right)$ is a cut for CP, that separates $\boldsymbol{d}^\star_{\mathcal{I}}$ from $\bar{\Delta}^{int}_{\mathcal{I}}$. Moreover, we have shown that any cut for CP can be obtained in this way. This enables us to develop a procedure based on the primal separation problem **P-SEP** – given $\boldsymbol{x}^0$ and $x^\star$, is there a valid inequality for $\mathcal{F}_{\text{SPP}}$, tight at $\boldsymbol{x}^0$, that separates $\boldsymbol{x}^\star$ from $\mathcal{F}_{\text{SPP}}$? If it exists, it will be transferred to CP to tighten the relaxation of $\bar{\Delta}^{int}_{\mathcal{I}}$. From the theoretical point of view, if $\boldsymbol{d}^\star$ is extremal, such a cut always exists as shown on Corollary 29.

Corollary 29. *Assume $\boldsymbol{d}^\star_{\mathcal{I}} \in Ext\,(\mathcal{F}_{\text{CP}})$, and let $\boldsymbol{d}^\star = \boldsymbol{T}\boldsymbol{d}^\star_{\mathcal{I}}$ (we still assume that $\boldsymbol{d}^\star_{\mathcal{I}}$ is not an integral direction). There always exists a valid inequality* (Γ) *tight at $\boldsymbol{x}^0$ that separates $\boldsymbol{x}^\star = \boldsymbol{x}^0 + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$ from $\mathcal{F}_{\text{SPP}}$.*

Proof. $\boldsymbol{d}^\star_{\mathcal{I}}$ is an extreme point of $\mathcal{F}_{\text{CP}}$ but does not belong to Δ^{int}. Since Δ^{int} is a finite subset of elements of $\mathcal{F}_{\text{CP}}$, and since $\mathcal{F}_{\text{CP}}$ is a polyhedron, there exists a cut $\left(\bar{\Gamma}\right)$ that separates $\boldsymbol{d}^\star_{\mathcal{I}}$ from $\bar{\Delta}^{int}_{\mathcal{I}}$. By Proposition 28, the result holds. □

It is also interesting to note that the linear transformation applied to the primal cut to transfer it to CP ($\bar{\boldsymbol{\alpha}}^T = \boldsymbol{\alpha}^T\boldsymbol{T}$) is the same as for the objective function ($\bar{\boldsymbol{c}}^T = \boldsymbol{c}^T\boldsymbol{T}$) and for the constraint matrix ($\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{I}} = \boldsymbol{A}_{\bar{\mathcal{P}}\boldsymbol{.}}\boldsymbol{T}$). Finally, note that adding cuts does not prevent to use the characterization of extreme integer solutions as those having a column-disjoint support. Since no cutting plane is added to CP that cuts any integer directions, the set of extreme integral directions $Ext\left(Conv\left(\bar{\Delta}^{int}_{\mathcal{I}}\right)\right)$ is still contained in the set of extreme directions $Ext\,(\mathcal{F}_{\text{CP}})$ even after the addition of cutting planes. Geometrically, adding any

valid inequality for $Conv\left(\bar{\Delta}_{\mathcal{I}}^{int}\right)$ to the relaxation cannot transform any nonextreme integral direction into a extreme one.

5.4.1 Specific Primal Separation Procedures

As exposed in the introduction, there exist previous works on the primal separation problem. It has been shown that **P-SEP** is equivalent to SPP in terms of complexity (Schulz et al., 1995). Also, there exist general-purpose families of primal cuts, such as Gomory–Young's cuts (Young, 1965). Iteratively adding cuts from these families would ultimately lead to a column-disjoint solution of CP, after a finite number of separations – provided that the choice of incoming variables follow some lexicographic order (Young, 1965). However, these families are not polyhedral, i.e., they do not take into account the SPP structure. We chose to study two families of inequalities that are known to yield strong cutting planes for SPP, namely Clique and Odd-cycle inequalities.

Primal Clique Inequalities

Let us consider the conflict graph of matrix $\boldsymbol{A}$, $\mathcal{G} = (\mathcal{N}, E)$, where each node of $\mathcal{N} = \{1, \ldots, n\}$ corresponds to a column of $\boldsymbol{A}$, and E is such that $\{i, j\} \in E$ iff $\boldsymbol{A}_{\cdot i}^T \boldsymbol{A}_{\cdot j} \neq 0$. Two vertices are linked by an edge if the corresponding columns are not disjoint. Given a clique $\mathcal{W}$ in this graph, any feasible solution of SPP satisfies

$$(\Gamma_{\mathcal{W}}) \; : \; \sum_{i \in \mathcal{W}} x_i \leq 1, \tag{5.10}$$

called the clique inequality associated with $\mathcal{W}$. Clique cuts form a family of generally strong cutting planes for SPP, and were first introduced by Padberg (1973). Given a fractional solution $\boldsymbol{x}^\star$ of SPP^{LR}, the standard clique separation problem consists in associating the weight $w_i = x_i^\star$ with each vertex of $\mathcal{G}$ and determine a clique of weight greater than 1 in $\mathcal{G}$ if any.

From the work of Letchford & Lodi (2003b), we will develop here a more efficient procedure for primal clique cuts. In the primal context, let $\boldsymbol{d}^\star$ be a fractional direction, and $\boldsymbol{x}^\star = \boldsymbol{x}^0 + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$. For $(\Gamma_{\mathcal{W}})$ to be tight at $\boldsymbol{x}^0$, we need $\sum_{i \in \mathcal{W}} x_i^0 = |\mathcal{W} \cap \mathcal{P}| = 1$. Hence, exactly one variable from $\mathcal{P}$ must be part of clique $\mathcal{W}$. Denote that variable as l, and the corresponding clique as $\mathcal{W}_l$. Furthermore, for $(\Gamma_{\mathcal{W}_l})$ to separate $\boldsymbol{x}^\star$ from $\mathcal{F}_{\text{SPP}}$, we must have $\sum_{i \in \mathcal{W}_l} x_i^\star > 1$. Since $x_i^\star = r(\boldsymbol{d}^\star) d_i^\star$ for all $i \in \mathcal{I}$, $\sum_{i \in \mathcal{W}_l} x_i^\star = x_l^\star + \sum_{i \in \mathcal{S}^+} x_i^\star$, where $\mathcal{S}^+ = Supp\left(\boldsymbol{d}_{\mathcal{I}}^\star\right)$. Denote also as $\mathcal{S}^- = Supp\left(\boldsymbol{d}_{\mathcal{P}}^\star\right)$ the set of columns of $\mathcal{P}$ that are impacted by direction $\boldsymbol{d}^\star$. Sets $(\mathcal{S}^-, \mathcal{S}^+)$ form a partition of $Supp\left(\boldsymbol{d}^\star\right)$ such that for all $i \in Supp\left(\boldsymbol{d}^\star\right)$, $i \in \mathcal{S}^+$ if $d_i^\star > 0$ and

$i \in \mathcal{S}^-$ if $d_i^\star < 0$, so

$$\sum_{i \in \mathcal{W}_l} x_i^\star > 1 \Leftrightarrow x_l^\star + \sum_{i \in \mathcal{W}_l \cap \mathcal{S}^+} x_i^\star > 1.$$

Because $\mathcal{W}_l$ is a clique in $\mathcal{G}$, then $\mathcal{W}_l \cap \mathcal{S}^+ \subseteq \mathcal{W}_l$ is also a clique. Therefore, there exists a primal clique inequality that separates $\boldsymbol{x}^\star$ from $\mathcal{F}_{\text{SPP}}$ iff for some $l \in \mathcal{S}^-$, there exists a clique $\mathcal{W}_l$ that satisfies (1) $\mathcal{W}_l \setminus \{l\} \subseteq N_l$, (2) $l \in \mathcal{W}_l$, and (3) $w(\mathcal{W}_l) = \sum_{i \in \mathcal{W}_l} x_i^\star > 1$, where N_l is the set of neighbors of l in $\mathcal{G}$ that are also in $\mathcal{S}^+$. These properties show that it is sufficient to look for a primal clique cut within $\mathcal{S}^+ \cup \mathcal{S}^-$ to solve the complete primal clique separation problem. Hence, the primal separation procedure for primal clique cuts, called `CL_PSEP` and summarized in Algorithm 4 returns an empty set of primal clique cuts if and only if none exists.

The algorithm always finds a cut if there exists one, and that will be the most violated. However, the size of the clique could be potentially increased by adding 0-weight variables. Even if Step 4 of Algorithm 4 requires solving a $\mathcal{NP}$-hard problem [2], note that the procedure is practically very fast because the size of N_l is usually small.

Algorithm 4: `CL_PSEP`

Input: $\boldsymbol{d}_{\mathcal{I}}^\star \leftarrow$ an optimal solution of CP (fractional direction).
Output: $\mathcal{K}$, a set of primal clique cuts (empty if none exists)

```
1 K ← ∅; d* ← T d*_I; S⁻ ← Supp(d*_P); S⁺ ← Supp(d*_I);
2 for l ∈ S⁻ do
3 |   G_l = (N_l, E_l) ← weighted subgraph of G induced by N_l ⊆ S⁺ (w_i = x*_i);
4 |   W_l ← clique of maximum weight in G_l ;
5 |   W_l ← W_l ∪ {l};
6 |   if w(W_l) > 1 then
7 |   |   K ← K ∪ {(Γ_{W_l})};
8 return K;
```

Primal Odd-Cycle Inequalities

Odd-cycle inequalities form another well-known family of valid inequalities for SPP. Given a cycle $\mathcal{Q}$ of odd length in $\mathcal{G}$, the following inequality

$$(\Gamma_{\mathcal{Q}}) \; : \; \sum_{i \in \mathcal{Q}} x_i \leq \frac{|Q| - 1}{2} \tag{5.11}$$

2. For the determination of the clique of maximal weight in $\mathcal{G}_l$, we use the *Cliquer* open source library, available at `http://users.aalto.fi/~pat/cliquer.html`, based on the algorithm described by Östergård (2001).

is valid for SPP. Clearly, $(\Gamma_{\mathcal{Q}})$ is tight at $\boldsymbol{x}^0$ if and only if $\sum_{i\in\mathcal{Q}} x_i^0 = |\mathcal{Q}\cap\mathcal{P}| = (|\mathcal{Q}|-1)/2$. Furthermore, since $\mathcal{P}$ is column-disjoint, there exists no edge between any pair of vertices of $\mathcal{P}$. Therefore, $\mathcal{Q}$ is an alternate cycle with vertices in $\mathcal{P}$ and $\mathcal{I}$, except for one $\mathcal{I}-\mathcal{I}$ edge (i.e., an edge that links two vertices of $\mathcal{I}$). Based on these observations and similarly to clique cuts, the search graph can be restricted to vertices in $Supp\,(\boldsymbol{d}^\star)$, but in a stronger manner

Proposition 30. *Every primal odd-cycle cut* $(\Gamma_{\mathcal{Q}})$ *that separates* $\boldsymbol{x}^\star$ *from* $\mathcal{F}_{\text{SPP}}$ *satisfies* $\mathcal{Q}\subseteq Supp\,(\boldsymbol{d}^\star)$.

Proof. Suppose that the result is false and that there exists a cycle $\mathcal{Q}\nsubseteq Supp\,(\boldsymbol{d}^\star)$. Let $\mathcal{S}^- = Supp\,(\boldsymbol{d}_{\mathcal{P}})$ and $\mathcal{S}^+ = Supp\,(\boldsymbol{d}_{\mathcal{I}})$. Define $\mathcal{Q} = (q_1, q_2, \ldots, q_{2T+1}, q_1)$, where $q_1, q_{2t+1}\in\mathcal{I}$ and $q_{2t}\in\mathcal{P}$ for all $t\in\{1,\ldots,T\}$. $\mathcal{Q}$ is an alternate cycle but for the $\{q_{2T+1}, q_1\}$ edge. Consider the two following cases:

(i) $\underline{q_1\notin\mathcal{S}^+}$. Then $d^\star_{q_1}=0$ and

$$\sum_{i\in\mathcal{Q}} x_i^\star = \sum_{i=2}^{2T+1} x^\star_{q_i} = \sum_{t=1}^{T}\left(x^\star_{q_{2t}} + x^\star_{q_{2t+1}}\right).$$

Every (q_{2t}, q_{2t+1}) is an edge of $\mathcal{G}$, so for every pair of columns $(\boldsymbol{A}_{\cdot 2t}, \boldsymbol{A}_{\cdot 2t+1})$ there exists a row i such that $A_{i(2t)} = A_{i(2t+1)} = 1$. The linear set partitioning constraint that corresponds to that conflict yields $x^\star_{q_{2t}} + x^\star_{q_{2t+1}}\leq 1$ for all $t\in\{1,\ldots T\}$. Hence, $\sum_{i\in\mathcal{Q}} x_i^\star \leq T = (|Q|-1)/2$ and $\boldsymbol{x}^\star$ does not violate $(\Gamma_{\mathcal{Q}})$.

(ii) $\underline{q_2\notin\mathcal{S}^-}$. From $\boldsymbol{A}\boldsymbol{d}^\star = \boldsymbol{0}$ and $\boldsymbol{d}^\star_{\mathcal{I}}\geq\boldsymbol{0}$, necessarily $q_1, q_3\notin\mathcal{S}^+$. Case (i) applies and this concludes the proof.

□

Proposition 30 suggests to distinguish between $\mathcal{S}^- - \mathcal{S}^+$ and $\mathcal{S}^+ - \mathcal{S}^+$ edges. Let $\mathcal{G}_B = (N_B, E_B)$ be a subgraph of $\mathcal{G}$ with $N_B = \mathcal{S}^-\cup\mathcal{S}^+ = Supp\,(\boldsymbol{d}^\star)$, and $\{i,j\}\in E_B$ iff $i\in\mathcal{S}^-$, $j\in\mathcal{S}^+$, and $\{i,j\}\in E$ (in conflict graph $\mathcal{G}$). $\mathcal{G}_B$ is a bipartite graph. In the case of the primal odd-cycle separation, the edges (and not the vertices) of the graph are weighted as $w_{ij} = 1 - x_i^\star - x_j^\star$ if $\{i,j\}\in E_B$. The weight of a path or a cycle is the sum of the weights of its edges. Given a cycle $\mathcal{Q}$, its weight is therefore $w(\mathcal{Q}) = |\mathcal{Q}| - 2\sum_{i\in\mathcal{Q}} x_i^\star$ and the corresponding odd-cycle inequality is violated by $\boldsymbol{x}^\star$ iff $w(\mathcal{Q})<1$. The primal odd-cycle separation procedure, `CY_PSEP` is given in Algorithm 5. This algorithm is usually fast because it consists of finding at most $|\mathcal{S}^-|$ shortest paths in a relatively small bipartite graph $\mathcal{G}_B$.

Algorithm 5: CY_PSEP

Input: $\boldsymbol{d}^\star_{\mathcal{I}} \leftarrow$ an optimal solution of CP (fractional direction).
Output: $\mathcal{K}$, a set of primal odd-cycle cuts (empty if none exists)

$\mathcal{K} \leftarrow \emptyset$; $\boldsymbol{d}^\star \leftarrow \boldsymbol{T}\boldsymbol{d}^\star_{\mathcal{I}}$; $\mathcal{S}^- \leftarrow \mathit{Supp}\,(\boldsymbol{d}^\star_{\mathcal{P}})$; $\mathcal{S}^+ \leftarrow \mathit{Supp}\,(\boldsymbol{d}^\star_{\mathcal{I}})$; $\boldsymbol{x}^\star \leftarrow \boldsymbol{x}^0 + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$;
Build weighted graph $\mathcal{G}_B$ ($w_{ij} = 1 - x^\star_i - x^\star_j$);
for $i, j \in \mathcal{S}^+$ *such that* $\{i, j\} \in E$ *($\mathcal{S}^+$–$\mathcal{S}^+$ conflict)* **do**
    $(q_1, q_2, \ldots, q_{2T}, q_{2T+1}) \leftarrow$ shortest path from i to j in $\mathcal{G}_B$ ($i = q_1$, $j = q_{2T+1}$);
    $\mathcal{Q} \leftarrow (q_1, q_2, \ldots, q_{2T}, q_{2T+1}, q_1)$ (odd-cycle);
    if $w(\mathcal{Q}) = w_{ij} + \sum_{k=1}^{2T+1} w_{q_k q_{k+1}} < 1$ **then**
        $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\Gamma_{\mathcal{Q}})\}$;
return $\mathcal{K}$;

5.4.2 Algorithm

Algorithm 6 incorporates the cutting plane results discussed in the previous sections into Algorithm 3. The different (nonexhaustive) branching strategies that we use are presented in Section 5.5.

SEP_MAX is a parameter that represents the maximum number of separation problems to be solved consecutively. Explicit values chosen for INC_MAX and SEP_MAX, as well as the methods to determine the set of variables to be fixed (line 33) are discussed in Section 5.5 before the numerical results are given. Different priority rules for choosing the separation algorithm (line 28) are studied and the corresponding results are displayed in the next section.

5.5 Numerical Results

The results presented in this section empirically show the relevance of our theoretical results. We show that primal cuts indeed improve the performance of our algorithm and help foster integral solutions in ISUD.

5.5.1 Methodology

Instances. The instances presented here are those used by Zaghrouti et al. (2014) (SPPAA01, VCS1200, and VCS1600) to which we added another instance from the OR-Library[3], SPPAA04. Both SPPAA01 and SPPAA04 are small flight assignment problems (respectively 823 constraints and 8,904 variables, and 426 constraints and 7,195 constraints) with an average of 9 nonzero entries per column. VCS1200 is a medium-size bus driver scheduling problem

3. http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Algorithm 6: `ISUD_CUTS`

Input: $\boldsymbol{x}^0$: a solution of SPP; INC_MAX: maximal incompatibility degree considered; SEP_MAX: maximal number of consecutive **P-SEP**;
Output: $\boldsymbol{x}^k$, a better solution of SPP.

$c \leftarrow 0$ (number of consecutive **P-SEP** solved);
$\iota \leftarrow 1$ (current maximum incompatibility degree);
`Pool` $\leftarrow \emptyset$ (pool of valid inequalities);
Compute $\mathcal{P}$ and $\mathcal{C}$ associated with $\boldsymbol{x}^0$; $k \leftarrow 0$;
while *true* **do**
  If necessary, update $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{I}^\iota$ associated with $\boldsymbol{x}^k$;
  if $z^\star_{\mathrm{RP}} < 0$ **then**
    $\boldsymbol{\delta}^i \leftarrow$ an optimal solution of RP ($\bar{c}_i < 0$, $i \in \mathcal{C}^k$);
    $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + \boldsymbol{\delta}^i$; $k \leftarrow k + 1$;
  else
    Build CP^ι anew (without cutting planes nor fixed variables);
    continue $\leftarrow$ true;
    while *continue = true* **do**
      If necessary, update $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{I}^\iota$ associated with $\boldsymbol{x}^k$, and CP^ι;
      if $z^\star_{\mathrm{CP}^\iota} < 0$ **then**
        $\boldsymbol{d}^\star_{\mathcal{I}} \leftarrow$ an optimal solution of CP^ι; $\boldsymbol{d}^\star \leftarrow \boldsymbol{T}\boldsymbol{d}^\star_{\mathcal{I}}$;
        if $\boldsymbol{d}^\star_{\mathcal{I}}$ *is column-disjoint* **then**
          $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$;
          $k \leftarrow k + 1$;
          continue $\leftarrow$ false;
        else
          $\boldsymbol{x}^\star \leftarrow \boldsymbol{x}^0 + r(\boldsymbol{d}^\star)\boldsymbol{d}^\star$;
          if $c <$ *SEP_MAX* **then**
            $c \leftarrow c + 1$;
            if `Pool` *contains primal cuts that separate* $\boldsymbol{x}^\star$ *from* $\mathcal{F}_{\mathrm{SPP}}$ **then**
              Transfer all these cuts to CP^ι;
            else
              $\mathcal{K} \leftarrow$ set of primal cuts generated with `CL_PSEP` and/or `CY_PSEP`;
              if $\mathcal{K} \neq \emptyset$ **then**
                Transfer every cut in $\mathcal{K}$ to CP^ι;
                `Pool` $\leftarrow$ `Pool` $\cup\, \mathcal{K}$;
          else
            Fix at least one variable of CP^ι to zero;
            $c \leftarrow 0$;
      else
        if $\iota <$ *INC_MAX* **then**
          $\iota \leftarrow \iota + 1$;
          $c \leftarrow 0$;
        else
          return $\boldsymbol{x}^k$;

(1,200 constraints, 133,000 variables), and VCS1600 is a large bus driver scheduling problem (1,600 constraints, 571,000 variables); both have an average of 40 nonzero entries per column. These numbers of nonzero entries per column are typical of aircrew and bus driver scheduling problems and do not vary much with the number of constraints. In scheduling instances, they typically correspond to the number of tasks to be completed by a worker in a given time span. The nonzeros correspond to the number of tasks per duty. The optimal solutions of the problems are known: SPPAA01 and SPPAA04 were solved with CPLEX 12.4 [4], and VCS1200 and VCS1600 were made with all columns generated during a column-generation process by GENCOL [5] whose optimal solution is known.

Initial Solutions. In scheduling applications, a column generally represents a sequence of tasks performed by an employee. Define the *primal information* of a solution as the percentage of consecutive tasks in that solution that are also consecutive in the optimal schedule. In practice, this quantity is not known precisely a priori, but the following two observations hold for industrial bus driver or aircrew scheduling problems. First, crews do not often change vehicles during their duties, and their schedules therefore have many consecutive tasks in common with those of the vehicle routes. Second, when the schedule is updated, e.g., because of unforeseen events, the reoptimized schedule usually has many pieces in common with the original schedule (companies generally add penalties in the objective function to discourage changes). Thus, many consecutive tasks from the initial *paths* (vehicle routes or the original schedule) remain consecutive in the optimal schedule. In bus driver scheduling problems, the initial solution that follows the bus routes typically contains 90% of primal information (VCS1200, VCS1600). In aircrew scheduling, the figure is generally around 75% (Zaghrouti et al., 2014) (SPPAA01, SPPAA04). Therefore, we chose to perturb the (known) optimal solutions to generate initial solutions that contain a similar level of primal information to that experienced in practice. These solutions are generally infeasible, so we assign the perturbed columns a high cost, as is done in companies for the schedules that follow the bus/airplane routes. In our perturbation method, the input parameter π is the percentage of columns of the optimal solution that will appear in the new initial solution. For SPPAA04 and SPPAA01, we generated initial solutions for $\pi = 10\%$, 15%, 20%, and 35%; for VCS1200 and VCS1600, we used $\pi = 20\%$, 35%, and 50%. These parameters were chosen so that the resulting primal information (given in Table 5.1) is consistent with the typical values. The initial gaps range from 50% to 80%, depending on the instance.

4. CPLEX is freely available for academic and research purposes under the IBM academic initiative: `http://www-03.ibm.com/ibm/university/academic`

5. GENCOL is a commercial software developed at the GERAD research center and now owned by the AD OPT company, a division of KRONOS.

Table 5.1 Percentage of primal information in the initial solutions of the benchmark.

			Primal Information				
Instance	m	n	$\pi = 10\%$	$\pi = 15\%$	$\pi = 20\%$	$\pi = 35\%$	$\pi = 50\%$
SPPAA01	803	8,904	71.5	75.6	80.0	-	-
SPPAA04	423	7,195	-	64.0	70.1	78.5	-
VCS1200	1,200	133,000	-	-	87.6	91.1	93.9
VCS1600	1,600	570,000	-	-	86.8	90.9	93.9

Cutting Planes Strategies. The tests were conducted for the following cutting planes strategies:

NONE: Without primal cuts;

CLIQUE: With primal clique cuts only;

CYCLE: With primal odd-cycle cuts only;

BOTH: Both aforementioned cut types are separated at every **P-SEP** step;

PRIO: Primal odd-cycle cuts are separated only if no primal clique cut is found.

Moreover, another parameter is included, which is the maximum number of separation problems solved before using another technique (such as branching). We analyzed the results for different values ranging from 40 to 40,000 (virtually infinite). In a very large majority of the cases, either no cut could be found before the 40^{th} **P-SEP** was solved, or the cutting planes yield an integral direction in less than 40 **P-SEP**. Hence, we fixed the maximum number of consecutive **P-SEP** to 40 and only present these results here.

Branching Strategies. We propose four different nonexhaustive branching techniques to improve the performance of the algorithm. They correspond to the decision made when no primal cut is found that cuts the current fractional direction.

NOBR: stop the algorithm;

LAST: all variables of the last fractional direction found are set to zero in CP until an augmentation is performed;

FIRST: all variables of the first direction found since the last augmentation or the last branching are set to zero in CP until an augmentation is performed;

COVER: a subproblem is solved to determine a small subset of $\mathcal{I}$ such that the support of each fractional solution found since the last augmentation or the last branching contains at least one element of this set. These variables are set to zero in CP, so that all the aforementioned fractional solutions become infeasible.

All these techniques were experimented, but we present detailed results for NOBR and FIRST only. The performance for the other two branching strategies are exposed more briefly. The variation in the results is not significant enough to make a detailed presentation of all four aforementioned strategies.

5.5.2 Results

All the tests were performed on a Linux PC with processors of 3.4 GHz. For each problem and each perturbation parameter π, 10 different instances (different initial solutions and corresponding artificial columns) are generated. Column ALGO indicates the cutting strategy used; BEST is the best of the four on each instance, i.e., the one with the smallest gap, and in the event of a tie, the shortest computational time to reach the best solution. All times are given in seconds.

The commercial solver CPLEX was tested on all the instances of the benchmark, with the initial solutions given as MIP-start. Instances SPPAA01 and SPPAA04 are solved to optimality in an average of 22.1 and 14.4 seconds, respectively. Within a time limit of one hour, provided the initial solutions as a warm start, CPLEX only slightly improves them on the instance VCS1200, never lowering the gap under 14% (average gap of 49.7%). Within that same time limit, it never improves any of the initial solutions given for VCS1600 at all (the average gap remains 73% as for the initial solutions). Note that, if the initial solution is not given, CPLEX only finds a feasible solution for 1 of 30 instances for both VCS1200 and VCS1600 within one hour, and that solution is of comparable cost to that of our initial solutions. Here, one should consider SPPAA01 and SPPAA04 as benchmark instances used for a detailed analysis of our algorithm and its behavior, and VCS1200 and VCS1600 as practical instances that we aim to solve within a few minutes.

Tables 5.2 and 5.3 show the results of our algorithm for all instances generated from SPPAA01, with associated branching strategies NOBR and LAST, for parameters $INC\_MAX = 10$, $SEP\_MAX = 40$. Other values were tested and gave very similar results. Columns π and ALGO display the perturbation degree of the instances and the chosen separation strategy, respectively. The next three columns display the number of instances (out of 10) that were solved to optimality (0%), with a positive gap $\leq 2\%$, and with a gap $> 2\%$; the mean gap is given in column MEAN. Then, the overall computation time (t_{ISUD}), the time to reach the best solution obtained (t_{BEST}) and the average time per augmentation ($t_{\mathbf{AUG}}$) are displayed. The last columns contain the mean number of augmentations K performed to reach the best solution, and the mean size of $|\mathcal{S}| = \mathit{Supp}(\boldsymbol{d})$ for disjoint ($|\mathcal{S}|^D$) and nondisjoint ($|\mathcal{S}|^N$) directions. Note that, while the other mean values are computed over all executions

Table 5.2 Comparison of the performances of ISUD with branching NOBR for instance SPPAA01. Parameters: $INC\_MAX = 10$ and $SEP\_MAX = 40$.

		GAP				Time (sec.)			**AUG**		
π	ALGO	0%	$\leq 2\%$	$> 2\%$	MEAN	t_{ISUD}	t_{BEST}	$t_{\mathbf{AUG}}$	K	$\lvert\mathcal{S}\rvert^D$	$\lvert\mathcal{S}\rvert^N$
10%	NONE	2	2	6	213.6%	6.2	5.1	0.21	29.	4.5	101.
	CLIQUE	3	5	2	72.1%	12.5	8.6	0.28	33.	4.9	122.
	CYCLE	3	2	5	197.5%	8.0	5.6	0.22	31.	4.6	143.
	BOTH	3	3	4	145.8%	15.9	7.4	0.25	35.	4.6	142.
	PRIO	3	5	2	72.1%	16.1	9.3	0.30	33.	4.9	113.
	BEST	**3**	**5**	**2**	**72.1%**	**11.7**	**7.9**	-	-	-	-
15%	NONE	2	3	5	134.1%	6.7	5.3	0.17	37.	3.7	48.
	CLIQUE	5	3	2	70.5%	10.4	6.7	0.18	39.	3.9	70.
	CYCLE	3	4	3	102.1%	8.0	5.9	0.18	37.	3.8	70.
	BOTH	5	4	1	41.4%	15.7	7.3	0.20	39.	4.0	95.
	PRIO	5	4	1	41.4%	15.6	7.4	0.20	39.	4.0	98.
	BEST	**5**	**4**	**1**	**41.4%**	**9.2**	**6.7**	-	-	-	-
20%	NONE	7	2	1	21.3%	7.6	6.0	0.17	37.	3.4	32.
	CLIQUE	5	5	0	0.2%	11.0	6.9	0.17	40.	3.4	57.
	CYCLE	6	3	1	40.7%	7.7	6.0	0.16	39.	3.4	43.
	BOTH	6	4	0	0.2%	15.0	7.0	0.18	40.	3.4	76.
	PRIO	6	4	0	0.2%	13.3	6.9	0.17	40.	3.4	74.
	BEST	**9**	**1**	**0**	**0.0%**	**10.4**	**6.3**	-	-	-	-

Table 5.3 Comparison of the performances of ISUD with branching LAST for instance SPPAA01. Parameters: $INC\_MAX = 10$ and $SEP\_MAX = 40$.

		GAP				Time (sec.)			**AUG**		
π	ALGO	0%	$\leq 2\%$	$> 2\%$	MEAN	t_{ISUD}	t_{BEST}	$t_{\mathbf{AUG}}$	K	$\lvert\mathcal{S}\rvert^D$	$\lvert\mathcal{S}\rvert^N$
10%	NONE	3	3	4	141.3%	11.4	8.7	0.27	38.	4.4	112.
	CLIQUE	3	3	4	80.2%	52.6	33.0	0.92	39.	4.7	198.
	CYCLE	3	2	5	139.0%	18.0	12.0	0.35	37.	4.3	151.
	BOTH	4	1	5	110.8%	76.2	32.2	0.91	40.	4.6	208.
	PRIO	5	1	4	79.1%	54.6	24.1	0.68	39.	4.7	177.
	BEST	**7**	**1**	**2**	**70.4%**	**33.2**	**19.4**	-	-	-	-
15%	NONE	2	4	4	30.9%	14.9	12.2	0.30	44.	4.0	87.
	CLIQUE	7	2	1	29.8%	28.2	9.6	0.24	41.	4.0	142.
	CYCLE	5	3	2	57.2%	16.0	10.6	0.27	41.	3.9	129.
	BOTH	8	2	0	0.0%	30.7	16.3	0.39	42.	4.1	111.
	PRIO	9	1	0	0.0%	34.6	17.6	0.43	41.	4.1	120.
	BEST	**9**	**1**	**0**	**0.0%**	**25.0**	**14.8**	-	-	-	-
20%	NONE	7	2	1	20.1%	9.9	7.0	0.17	42.	3.3	67.
	CLIQUE	6	4	0	0.0%	22.2	9.2	0.21	43.	3.5	104.
	CYCLE	8	2	0	0.1%	12.7	8.5	0.20	42.	3.4	86.
	BOTH	9	1	0	0.0%	23.6	9.8	0.22	45.	3.6	91.
	PRIO	9	1	0	0.0%	24.4	9.7	0.21	45.	3.6	102.
	BEST	**9**	**1**	**0**	**0.0%**	**16.1**	**9.1**	-	-	-	-

of the algorithm, the mean number of augmentations K is based on the instances for which the final gap is $\leq 2\%$. Large gaps often mean a much smaller number of iterations, and

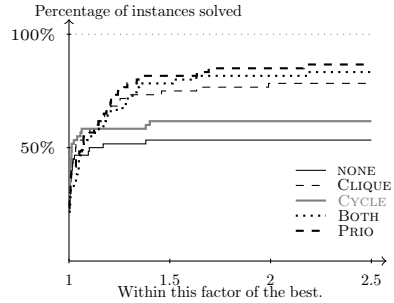
Table 5.4 Comparison of the performances of ISUD with branching NOBR for instance SPPAA04. Parameters: $INC\_MAX = 10$ and $SEP\_MAX = 40$.

		GAP				Time (sec.)			**AUG**		
π	ALGO	0%	$\leq 2\%$	$> 2\%$	MEAN	t_{ISUD}	t_{BEST}	$t_{\mathbf{AUG}}$	K	$\lvert\mathcal{S}\rvert^D$	$\lvert\mathcal{S}\rvert^N$
15%	NONE	1	0	9	518.8%	2.6	1.8	0.10	29.	3.4	86.
	CLIQUE	3	2	5	199.1%	8.1	3.3	0.14	26.	4.0	159.
	CYCLE	2	1	7	388.9%	3.7	2.3	0.12	26.	3.9	102.
	BOTH	5	2	3	160.8%	12.4	4.1	0.16	30.	4.1	154.
	PRIO	5	2	3	160.8%	9.8	4.0	0.16	30.	4.1	148.
	BEST	**5**	**2**	**3**	**160.8%**	**6.9**	**3.8**	-	-	-	-
20%	NONE	4	0	6	202.5%	3.2	2.2	0.10	28.	3.4	60.
	CLIQUE	5	2	3	83.0%	6.6	3.0	0.12	28.	3.7	138.
	CYCLE	4	0	6	202.5%	3.5	2.2	0.10	28.	3.4	93.
	BOTH	7	2	1	0.7%	11.6	3.7	0.14	27.	3.8	150.
	PRIO	7	2	1	0.7%	9.1	3.6	0.13	27.	3.8	143.
	BEST	**8**	**1**	**1**	**0.6%**	**5.9**	**3.2**	-	-	-	-
35%	NONE	9	0	1	0.6%	3.3	2.2	0.9	24.	3.0	73.
	CLIQUE	9	0	1	0.6%	7.1	2.2	0.9	25.	2.9	153.
	CYCLE	9	0	1	0.6%	3.8	2.2	0.9	25.	2.9	117.
	BOTH	9	0	1	0.6%	10.9	2.2	0.9	25.	2.9	162.
	PRIO	9	0	1	0.6%	8.1	2.2	0.9	25.	2.9	155.
	BEST	**9**	**0**	**1**	**0.6%**	**4.2**	**2.1**	-	-	-	-

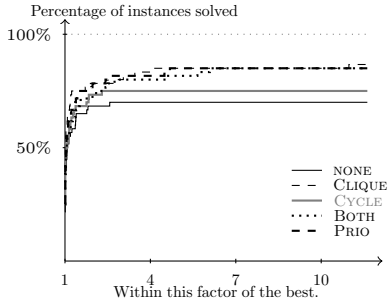
Table 5.5 Comparison of the performances of ISUD with branching LAST for instance SPPAA04. Parameters: $INC\_MAX = 10$ and $SEP\_MAX = 40$.

		GAP				Time (sec.)			**AUG**		
π	ALGO	0%	$\leq 2\%$	$> 2\%$	MEAN	t_{ISUD}	t_{BEST}	$t_{\mathbf{AUG}}$	K	$\lvert\mathcal{S}\rvert^D$	$\lvert\mathcal{S}\rvert^N$
15%	NONE	3	3	4	242.3%	6.2	4.9	0.18	31.	4.0	72.
	CLIQUE	4	3	3	189.3%	34.9	20.5	0.72	31.	4.1	180.
	CYCLE	3	3	4	211.5%	7.2	4.9	0.18	30.	3.9	102.
	BOTH	5	2	3	157.1%	34.3	13.9	0.48	31.	4.1	177.
	PRIO	5	2	3	157.1%	24.3	10.9	0.38	31.	4.1	168.
	BEST	**6**	**1**	**3**	**151.7%**	**24.3**	**15.0**	-	-	-	-
20%	NONE	3	2	5	224.3%	5.4	3.6	0.13	33.	3.4	76.
	CLIQUE	7	3	0	0.2%	15.6	9.7	0.30	32.	4.0	146.
	CYCLE	3	3	4	215.2%	9.0	5.2	0.18	35.	3.4	110.
	BOTH	9	0	1	48.3%	28.7	10.8	0.35	32.	3.9	150.
	PRIO	7	1	2	46.9%	27.8	14.0	0.47	31.	3.8	142.
	BEST	**10**	**0**	**0**	**0.0%**	**15.1**	**7.9**	-	-	-	-
35%	NONE	9	1	0	0.0%	3.5	2.4	0.9	25.	2.9	65.
	CLIQUE	9	1	0	0.1%	8.9	3.0	0.11	27.	3.0	140.
	CYCLE	9	1	0	0.0%	4.3	2.5	0.10	25.	2.9	108.
	BOTH	10	0	0	0.0%	12.9	2.5	0.10	27.	2.9	158.
	PRIO	10	0	0	0.0%	9.8	2.5	0.9	26.	2.9	150.
	BEST	**10**	**0**	**0**	**0.0%**	**5.3**	**2.4**	-	-	-	-

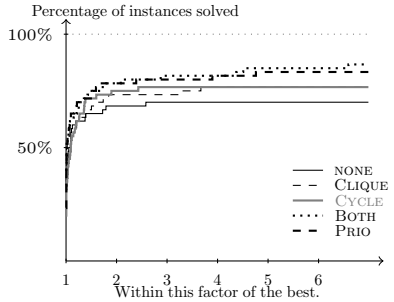
taking these instances into account would distort that mean. Tables 5.4 and 5.5 show the results of the same experiments for instances generated from SPPAA04.



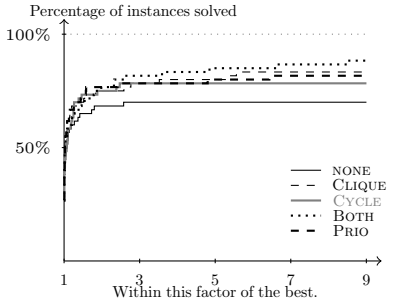
(a) Branching strategy: NOBR.



(b) Branching strategy: LAST.



(c) Branching strategy: FIRST.



(d) Branching strategy: COVER.

Figure 5.6 Performance diagrams over all SPPAA04 and SPPAA01 instances for the four branching strategies (NOBR (top-left), LAST (top-right), FIRST (bottom-left) and COVER (bottom-right). An instance is considered as solved if the gap is lower than 2%.

Consistently with our expectations and whatever the cutting planes or branching strategy, the primal information strongly influences the results of the algorithm. Namely, the higher the percentage of primal information is, the better the algorithm performs, both in terms of quality of the solution, and average running time. The branching strategies highlight common features and global differences between the various cutting techniques. First, the performance of the algorithms can be globally ranked from worst to best as follows: NONE $\preceq$ CYCLE $\preceq$ CLIQUE $\preceq$ PRIO $\preceq$ BOTH. Moreover, the execution time is significantly shorter for NONE and CYCLE than for the others. In the case of NONE, no primal separation problem is solved, and either no branching, or very simple fixing rules are applied, hence the high speed of the algorithm. In the case of CYCLE, the number of primal cycle cuts found is quite small, so the computation time is also smaller. The same holds for the time spent to reach the best solution (t_{BEST}) and the time per augmentation ($t_{\mathbf{AUG}}$). Note that the difference is much bigger for t_{ISUD} than for t_{BEST} because the time spent at the optimal solution to generate cutting planes is important. The number of augmentation steps is higher for the algorithms that generate more primal cuts (column K). Generally, cutting planes allow the algorithm to find more disjoint solutions within the same phase, hence yielding a larger number of

steps for each incompatibility degree. The algorithms using cuts tend to generate disjoint and nondisjoint combinations of larger size (columns $|\mathcal{S}|^D$ and $|\mathcal{S}|^N$). Cutting planes tend to make the problem more complex, add constraints, and the size of the support of a basic solution of CP therefore increases with the number of cuts inserted, hence the larger size of the combinations.

Furthermore, the algorithm is significantly better if fixing variables is available. This is particularly true when no cut is applied. Both nonexhaustive branching and cutting planes improve the performance of the algorithm. If we analyze the performance over SPPAA01 for $\pi = 15\%$ (closest to real-life problems) we see that (1) when no branching is made, PRIO solves 5/10 problems to optimality (against 2/10 for NONE), and 9/10 within 2% of the optimum (5/10 for NONE); and (2) when variables are fixed, PRIO solves 9/10 problems to optimality (2/10 for NONE) and all of them within 2% of the optimum (6/10 for NONE). Cutting planes therefore solve 7 out of the 8 problems that ISUD did not solve previously, hence yielding an improvement of 87.5% over SPPAA01 for $\pi = 15\%$.

Figure 5.6 displays the four performance diagrams of the algorithms for branching strategies NOBR, LAST, FIRST and COVER, over all SPPAA04 and SPPAA01 instances (70). Here again, whatever the branching strategy, cutting planes allow to solve many more instances. However, the improvement factor is much higher when no variable is fixed (less that 50% of the instances are solved without cuts; against more than 80% for PRIO or BOTH separation strategies). Interestingly, the time-increase factor (x-axis) never exceeds 10, and is most of the time lower than 4. Hence, the addition of cutting planes does not slow down the algorithm too much. Detailed computing times for each of the two problems are displayed in Figures 5.7 and 5.8, respectively. The COVER branching strategy shows slightly better performances, but not significantly enough to draw any conclusion yet. Moreover, more instances are solved by ISUD faster than by CPLEX.

Tables 5.6 and 5.7 display specific characteristics concerning the cutting planes generated during the process. For each branching strategy (NOBR and LAST), the number of instances solved within 2% of the optimum is shown under label INST. The mean time spent in the primal separation and cut management process including the cut pool management is given (t_{SEP}), as well as the mean number of separation problems solved (n_{SEP}), the mean number of primal clique cuts (n_{CL}), and primal odd-cycle cuts (n_{CY}). We can see that the time spent in the separation process is much higher when branching is applied. Indeed, in our algorithm, between two branching stages, SEP_MAX separation problems are solved. When no branching is performed, at most one sequence of SEP_MAX separation problems are solved and in case of failure to find a new primal cut, the algorithm instantly stops. This

Table 5.6 Cutting planes behavior of ISUD on SPPAA01. Comparison of branching strategies NOBR and LAST.

		INST		t_{SEP} (sec.)		MEAN					
						NOBR			LAST		
GAP	ALGO	NOBR	LAST	NOBR	LAST	n_{SEP}	n_{CL}	n_{CY}	n_{SEP}	n_{CL}	n_{CY}
$\leq 2\%$	NONE	18	21	0.0	0.0	0	0	0	0	0	0
	CLIQUE	26	25	0.5	2.8	56	240	0	164	853	0
	CYCLE	21	23	0.3	0.7	12	0	28	44	0	125
	BOTH	25	25	1.7	7.1	74	289	95	159	808	233
	PRIO	27	26	1.2	4.9	95	356	22	219	1073	48
$> 2\%$	NONE	12	9	0.0	0.0	0	0	0	0	0	0
	CLIQUE	4	5	1.5	29.9	68	454	0	290	3383	0
	CYCLE	9	7	1.1	6.8	21	0	122	90	0	424
	BOTH	5	5	7.1	51.8	59	653	202	241	2957	567
	PRIO	3	4	4.8	27.3	141	998	79	336	3259	153

Table 5.7 Cutting planes behavior of ISUD on SPPAA04. Comparison of branching strategies NOBR and LAST.

		INST		t_{SEP} (sec.)		MEAN					
						NOBR			LAST		
GAP	ALGO	NOBR	LAST	NOBR	LAST	n_{SEP}	n_{CL}	n_{CY}	n_{SEP}	n_{CL}	n_{CY}
$\leq 2\%$	NONE	14	21	0.0	0.0	0	0	0	0	0	0
	CLIQUE	21	27	1.5	3.1	83	446	0	149	734	0
	CYCLE	16	22	0.3	0.5	15	0	52	29	0	100
	BOTH	25	26	5.2	8.4	91	438	103	138	665	165
	PRIO	25	25	2.2	3.2	106	480	31	144	658	44
$> 2\%$	NONE	16	9	0.0	0.0	0	0	0	0	0	0
	CLIQUE	9	3	1.6	34.9	58	366	0	540	3997	0
	CYCLE	14	8	0.4	3.3	17	0	39	101	0	514
	BOTH	5	4	5.2	41.5	74	426	88	375	2458	580
	PRIO	5	5	2.4	25.3	95	453	30	576	3033	216

can also be seen in the n_{SEP} column, for which the number of separation problems solved with branching is significantly higher than without branching. Moreover, the time t_{SEP}, and number of **P-SEP** n_{SEP} are significantly higher when the algorithm fails to find a good solution (gap $> 2\%$). This reflects the struggling of the algorithm to improve the solution and the associated running time increase. As it is often the case for the SPP, clique cuts are more efficient, and easier to find. Furthermore, as seen in Section 5.4.1, the requirements for an odd-cycle cut to be a primal cut are harder to meet (alternated cycles) than those that apply to clique cuts (only one vertex of the clique must be in the current solution); this also make them rarer.

Definition 9. A solution $\boldsymbol{x}^0$ is *ι-optimal* if it is optimal for the restriction of SPP to $\mathcal{P} \cup \mathcal{I}^\iota$, i.e., for the set of all at most ι-incompatible columns of $\boldsymbol{A}$ computed at $\boldsymbol{x}^0$.

Detailed results of ISUD on VCS1200 and VCS1600 are displayed in Table 5.8 and 5.9. Branch-

ing did not change the results, hence the figures shown here correspond to the NOBR strategy and the time limit of half an hour was never reached. The first column shows the perturbation degree π, and the second the cutting plane strategy ALGO. The next four display the number of instances solved to optimality (0%), with a positive gap $\leq 2\%$ and with a gap $> 2\%$, as well as the mean gap (MEAN). Columns t_{ISUD} and t_{BEST} indicate the total running time and the time spent before reaching the best solution found, respectively. Then, the number of instances for which ι-optimality has been proved is given for $\iota = 7$ and $\iota = 8$, and finally the mean size of the disjoint ($|\mathcal{S}|^D$) and nondisjoint ($|\mathcal{S}|^N$) combination are shown.

Table 5.8 Results for instance VCS1200. Parameters: $INC\_MAX = 8$ and $SEP\_MAX = 40$; Branching NOBR.

		GAP				Time (sec.)		ι-OPT		**AUG**	
π	ALGO	0%	$\leq 2\%$	$> 2\%$	MEAN	t_{ISUD}	t_{BEST}	7-OPT	8-OPT	$\lvert\mathcal{S}\rvert^D$	$\lvert\mathcal{S}\rvert^N$
20%	NONE	4	0	6	20.3%	53	46	3	2	2.5	198
	CLIQUE	4	0	6	20.3%	112	46	8	2	2.5	480
	CYCLE	4	0	6	20.3%	55	46	3	2	2.5	213
	BOTH	4	0	6	20.3%	164	46	8	2	2.5	476
	PRIO	4	0	6	20.3%	112	46	8	2	2.5	480
35%	NONE	8	0	2	2.3%	50	43	0	0	2.3	151
	CLIQUE	8	0	2	2.3%	142	55	8	0	2.3	361
	CYCLE	8	0	2	2.3%	53	43	0	0	2.3	176
	BOTH	8	0	2	2.3%	166	49	8	0	2.3	371
	PRIO	8	0	2	2.3%	142	55	8	0	2.3	361
50%	NONE	10	0	0	0.0%	37	28	0	0	2.2	132
	CLIQUE	10	0	0	0.0%	81	29	10	0	2.2	315
	CYCLE	10	0	0	0.0%	39	29	0	0	2.2	156
	BOTH	10	0	0	0.0%	130	30	10	0	2.2	354
	PRIO	10	0	0	0.0%	81	29	10	0	2.2	315

In the experiments on VCS1200 and VCS1600, we chose a lower value for the maximum incompatibility number (INC_MAX= 8) than for the smaller problems; this choice is motivated by several reasons. First, it limits the running time of the algorithm. Second, this number is already high compared to what swap heuristics can consider. As explained in Section 5.3.1, the incompatibility degree of $\boldsymbol{A}_{\cdot j}$ is proportional to the number of sequences of consecutive tasks from $\boldsymbol{A}_{\cdot j}$ that are not performed consecutively in the current solution. Hence, it is a kind of measure of the primal distance between a column and the current solution. This number can be compared to the typical parameter of a swap heuristic that tries to swap parts of the current solution to form new individual schedules which is the number of times an individual schedule of the current solution may be split. Nonbasic columns for which $\iota = 8$ are made of at least 4 separate sequences of tasks performed consecutively in the current solution (there is no maximum number); in practice, this number usually ranges between 6 and 7. For an average of 40 nonzero entries per columns, this number therefore seems reasonable. It is in particular much higher than the typical number of splits allowed in

Table 5.9 Results for instance VCS1600. Parameters: $INC\_MAX = 8$ and $SEP\_MAX = 40$; Branching NOBR.

		GAP				Time (sec.)		ι-OPT		**AUG**	
π	ALGO	0%	$\leq 2\%$	$> 2\%$	MEAN	t_{ISUD}	t_{BEST}	7-OPT	8-OPT	$\lvert\mathcal{S}\rvert^D$	$\lvert\mathcal{S}\rvert^N$
20%	NONE	5	0	5	16.5%	462	324	6	6	2.8	528
	CLIQUE	5	0	5	16.5%	646	325	6	6	2.8	831
	CYCLE	5	0	5	16.5%	465	324	6	6	2.8	534
	BOTH	5	0	5	16.5%	723	355	6	6	2.8	806
	PRIO	5	0	5	16.5%	647	325	6	6	2.8	831
35%	NONE	6	0	4	5.3%	433	244	7	6	2.4	519
	CLIQUE	6	0	4	5.3%	750	244	9	6	2.4	753
	CYCLE	6	0	4	5.3%	459	244	7	6	2.4	539
	BOTH	6	0	4	5.3%	813	244	9	6	2.4	761
	PRIO	6	0	4	5.3%	750	244	9	6	2.4	753
50%	NONE	8	0	2	2.4%	404	156	9	8	2.2	517
	CLIQUE	8	0	2	2.4%	492	156	9	8	2.2	691
	CYCLE	8	0	2	2.4%	405	156	9	8	2.2	521
	BOTH	8	0	2	2.4%	521	156	9	8	2.2	693
	PRIO	8	0	2	2.4%	492	156	9	8	2.2	691

the existing schedules in a swap heuristic, and the neighborhood explored by our algorithm is hence significantly wider than that of a swap heuristic.

Furthermore, the numerical results show that, even though primal cuts yield no improvement, they prove the ι-optimality of the solution in many cases, i.e., they show that the solution returned by ISUD is optimal for the restriction of SPP to the set of at most ι-incompatible columns ($\mathcal{I}^\iota$). This is especially true in the case of VCS1200, where 3 instances are proven 7-optimal without cutting planes, whereas 26 are with cutting planes.

Finally, the failure of the cutting planes to improve the results of ISUD on VCS1200 and VCS1600 can be partly explained by the nature of these instances. Although they describe real-world problems, they are in fact large instances obtained from column generation solution of bus drivers scheduling instances. All columns generated throughout the Branch-and-Price process are stored and put together to form a large scale instance, the solution of which is known. However, due to the intrinsic nature of that solution process, the density of the vertices of the resulting relaxed polyhedron ($\mathcal{F}_{\text{SPP}^{LR}}$) tends to be higher near the optimal solution. Hence, if a wrong decision is made in the beginning, there is a high probability of going to a region of the polyhedron where very few extreme integer neighbors, and thus integer directions, exist. Getting back to an area where the density of integer neighbors is higher, only by traveling alongside integer-to-integer edges, may then become extremely complicated. In the context of all-integer column generation, i.e., alternatively generate columns and improve the integer solution with ISUD as evoked in the introduction, this issue would disappear, because the dynamically generated columns increase the density of ex-

treme points around the current solution, hence providing many more potential directions to the complementary problem.

5.6 Conclusions

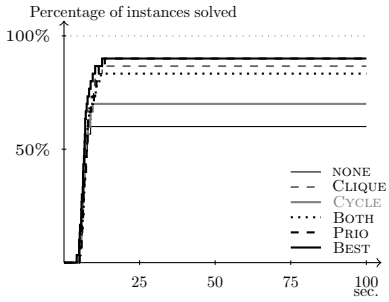
In this work, we proposed a purely primal formulation of ISUD and introduced cutting planes in the complementary problem CP. We also presented the algorithm as a matheuristic, i.e., the incompatibility degree defines a neighborhood of the current solution on which the augmentation problem is solved with integer programming techniques. The efficiency of linear programming and of our separation algorithms (Algorithms 4 and 5) allow us to explore a larger neighborhood than what a typical exchange heuristic would consider.

The work conducted on the mathematical formulation led us to characterize the set of cuts that can be transferred to CP as a nonempty subset of primal cuts tight at $\boldsymbol{x}^0$. We showed that, given a fractional direction $\boldsymbol{d}^\star$, a primal clique (or odd-cycle) cut exists if and only if there exists one that only involves variables of $Supp\,(\boldsymbol{d}^\star)$. Furthermore, in the case of primal odd-cycles, the separation over all variables is strictly equivalent to that over $Supp\,(\boldsymbol{d}^\star)$. Tests conducted over a benchmark of practical-like instances proved the potential of our method and highlight the importance of primal cutting planes in ISUD.

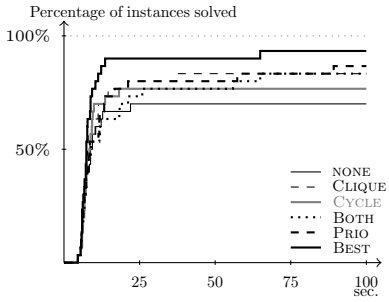
This work should be extended threefold. First, to gain better understanding of our algorithm, a larger benchmark is to be taken in consideration and a combination of cutting planes and other techniques (such as those proposed by Rosat et al. (2015a) for instance) must be tested over that larger set of problems. Second, other cutting planes families should be taken in consideration, and the corresponding primal separation procedures must be developed. Last, the algorithm need to be extended to more general {0,1}-programs, and not only to SPP instances. This last point seems to be the most important if the method is to be used in practice, since supplementary constraints are often added to set partitioning problems. All these extensions are active research subjects and the present work will have extensions in a near future.

Acknowledgements

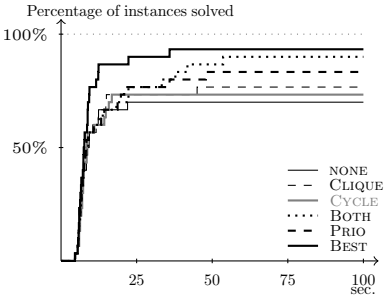
This work was supported by a Collaborative Research and Development grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Kronos Inc. Samuel Rosat benefitted of a grant from the International Internship Program of the Fonds de Recherche du Québec Nature et Technologies (FRQNT).



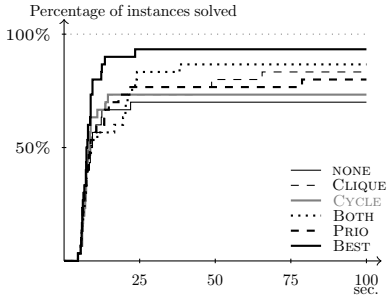
(a) Branching strategy: NOBR.



(b) Branching strategy: LAST.

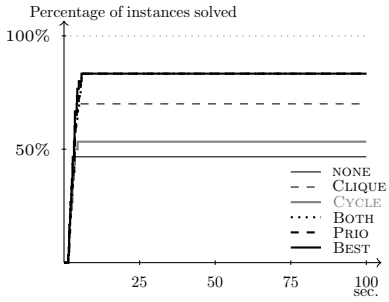


(c) Branching strategy: FIRST.

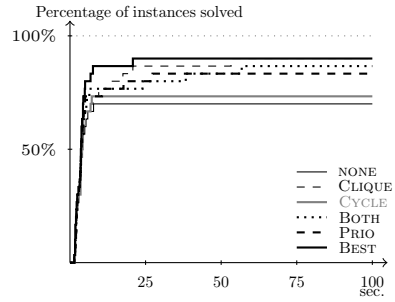


(d) Branching strategy: COVER.

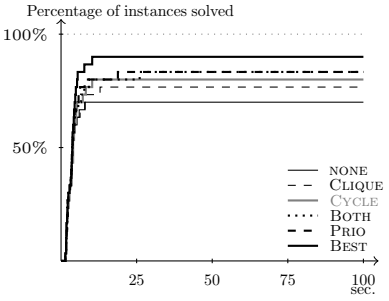
Figure 5.7 Percentage of instances solved over solution time for all SPPAA01 instances for the four branching strategies (NOBR (top-left), LAST (top-right), FIRST (bottom-left) and COVER (bottom-right). An instance is considered as solved if the gap is lower than 2%.



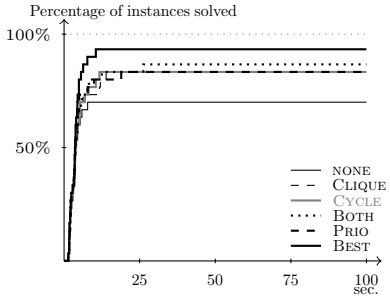
(a) Branching strategy: NOBR.



(b) Branching strategy: LAST.



(c) Branching strategy: FIRST.



(d) Branching strategy: COVER.

Figure 5.8 Percentage of instances solved over solution time for all SPPAA04 instances for the four branching strategies (NOBR (top-left), LAST (top-right), FIRST (bottom-left) and COVER (bottom-right). An instance is considered as solved if the gap is lower than 2%.

CHAPITRE 6 ARTICLE 3: DYNAMIC PENALIZATION OF FRACTIONAL DIRECTIONS IN THE INTEGRAL SIMPLEX USING DECOMPOSITION: APPLICATION TO AIRCREW SCHEDULING

Le texte de ce chapitre est celui de l'article *Dynamic penalization of fractional directions in the integral simplex using decomposition: Application to aircrew scheduling* publié dans les cahiers du GERAD et soumis à *European Journal of Operational Research* (Rosat et al., 2016). Auteurs : Samuel Rosat, Frédéric Quesnel, François Soumis, Issmail Elhallaoui.

Abstract

To solve integer linear programs, primal algorithms follow an augmenting sequence of integer solutions leading to an optimal solution. In this work, we focus on a particular primal algorithm, the integral simplex using decomposition (ISUD). To find the next point, one solves a linear program to select an augmenting direction for the current point from a cone of feasible directions. To ensure that this linear program is bounded, a normalization constraint is added and the optimization is performed on a section of the cone. The solution of the linear program, i.e., the direction proposed by the algorithm, strongly depends on the chosen normalization weights, and so does the likelihood that the next solution is integer. We modify ISUD so that the normalization is dynamically updated whenever the direction leads to a fractional solution, to penalize that direction. We propose several update strategies, based on theoretical and experimental results. To prove the efficiency of our strategies, we show that our version of the algorithm yields better results than the former version and than classical branch-and-bound techniques on a benchmark of industrial aircrew scheduling instances. The benchmark that we propose here is, to the best of our knowledge, comparable to no other from the literature. It provides large-scale instances with up to 1,700 flights and 115,000 pairings, hence as many constraints and variables, and the instances are given in a set-partitioning form together with initial solutions that accurately mimic those of industrial applications. Our work shows the strong potential of primal algorithms for the crew scheduling problem, which is a key challenge for large airlines.

6.1 Introduction

Crew pairing is a key challenge for large airlines: it is both financially significant and notably hard to solve. The fundamental challenge is the size of the instances: large airlines have grown continually since the very beginning of passenger air traffic, as a result of increases

in passenger volumes and occasional mergers (Continental Airlines and United Airlines, Air France and KLM, etc.). The problem involves determining a set of pairings (where a pairing is a sequence of flights and layovers that starts and ends at the same base) that covers all the scheduled flights at minimal cost over the planning horizon. The standard solution techniques are all based on the branch-and-bound algorithm. First, the integrality constraints are relaxed and an optimal fractional solution is found. Second, a branching tree is explored until integrality is restored and the integrality gap reduced to a given threshold. The exploration of the tree involves solving numerous linear relaxations of slightly modified versions of the original problem. The main drawbacks of this strategy are as follows. The size of the tree, and therefore the solution time, grows exponentially with the size of the data. Furthermore, the method used to solve the linear relaxations, the simplex algorithm, performs poorly on degenerate instances. An instance is *degenerate* when too many of the constraints are simultaneously saturated or, equivalently, some variables in the simplex basis have the value zero in most of the basic solutions. Crew scheduling problems have a high proportion of so-called set-partitioning constraints, and these increase the degeneracy. Moreover, it might take a long time to find an integer solution.

Another approach for these problems (and for integer linear programming in general) uses a *primal*, or *augmentation*, algorithm: it starts from a known feasible solution and iteratively improves it until optimality is reached. At each step it solves an augmentation subproblem that either furnishes an improving direction or asserts that the current solution is optimal. In the former case, this direction is followed from the current solution to a strictly better one. The main drawbacks of this method are the need for an initial feasible solution and the need to ensure that the improved solution is still integer. In the case of aircrew scheduling, the determination of good initial solutions proves to be relatively easy (see Section 6.5). The advantages of this approach are that it takes advantage of existing solutions, which is not possible with branch-and-bound, and it can quickly improve the given solution. The former is particularly important since it allows the approach to be used both for planning and for reoptimization after unforeseen events, when the existing solution can be used as a starting point.

This paper is organized as follows. In Section 6.2, we give an overview of augmentation methods, particularly in the context of the set partitioning problem (SPP) where these algorithms can be based on simplex pivots and are hence called *integral simplex* methods. In Section 6.3, we describe a specific augmentation algorithm, the integral simplex using decomposition (ISUD), based on work by Zaghrouti et al. (2014) and Rosat et al. (2015a). In Section 6.4 we improve this method by dynamically modifying one of the constraints in the augmentation subproblem. Four modification strategies are proposed, some based on

mathematical properties and others on experimental findings. The dynamic update aims to increase the likelihood that the augmentation subproblem returns an augmenting direction that leads to an integral solution; it is an extension of the work of Rosat et al. (2015a). In Section 6.5, we propose a new benchmark of SPPs generated from aircrew scheduling applications. This benchmark is an important contribution of the present work because it mimics as realistically as possible both the scheduling instances and the initial feasible solutions of real applications. To the best of our knowledge, no similar benchmark exists in the literature. In Section 6.6, we report numerical results that demonstrate the improvements obtained by our dynamic update of the constraints. Section 6.7 provides concluding remarks.

6.2 Integral simplex algorithms for the set partitioning problem

We assume that the reader is familiar with integer linear programming (see for example Schrijver (1998)). Let ILP denote a generic integer linear program. A *primal* (or *augmentation*) algorithm is a method that, given an initial ILP solution, outputs a sequence of feasible (integer) ILP solutions of strictly augmenting cost such that the final solution is optimal for ILP. By *augmenting*, we mean increasing for a maximization problem and decreasing for a minimization problem. In this paper, we consider minimization but the ideas also apply to maximization. Every primal method is based on the iterative solution of the *integral augmentation problem*, which can be stated as follows:

iAUG	Find an augmenting vector $\boldsymbol{z} \in \mathbb{Z}^n$ such that $(\boldsymbol{x} + \boldsymbol{z})$ is feasible for ILP and $\boldsymbol{z}$ is of negative cost, or assert that $\boldsymbol{x}$ is optimal for ILP.

Here $\mathbb{Z}^n$ is the set of integers, and $\boldsymbol{x}$ is a (known) solution of ILP. If an augmenting vector $\boldsymbol{z}$ is found, $\boldsymbol{x}' = (\boldsymbol{x} + \boldsymbol{z})$ becomes the next solution in the sequence, and **iAUG** is then solved again with $\boldsymbol{x}'$ as the initial solution. The process stops when $\boldsymbol{x}$ is determined to be optimal. Unfortunately, in general, the theoretical complexity of a single augmentation step is the same as that of solving the integer linear program (Schulz et al., 1995), and integer linear programming is $\mathcal{NP}$-hard. However, in practice, **iAUG** or a good relaxation of **iAUG** is relatively easy to solve and often produces integral augmentations on specific problems, including the one we consider here. For a detailed review of primal algorithms, see Spille & Weismantel (2005).

We concentrate on a specific integer linear program, the SPP, because it is the core of

scheduling models. The formulation is

$$z^\star_{\text{SPP}} = \min_{\boldsymbol{x}\in\mathbb{R}^n} \left\{ \boldsymbol{c}^T\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{e}\,,\ \boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{e}\,,\ \boldsymbol{x} \text{ is integer} \right\}, \tag{SPP}$$

where $\boldsymbol{A} \in \{0,1\}^{m\times n}$ is an $m \times n$ binary matrix, and $\boldsymbol{c} \in \mathbb{N}^n$ is the cost vector. Here $\boldsymbol{0}$ and $\boldsymbol{e}$ are vectors of zeroes and ones with size dictated by the context. Without loss of generality, we assume that $\boldsymbol{A}$ is full rank, contains no zero rows or columns, and has no identical rows or columns. The equalities $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{e}$ are called the linear constraints and $\boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{e}$ are the bound constraints. The set of all feasible solutions of SPP is denoted $\mathcal{F}_{\text{SPP}}$; the optimal value (or optimum) is $z^\star_{\text{SPP}}$; and any feasible solution $\boldsymbol{x}^\star \in \mathcal{F}_{\text{SPP}}$ such that $\boldsymbol{c}^T\boldsymbol{x}^\star = z^\star_{\text{SPP}}$ is an optimal solution of SPP. Because of the bounds ($\boldsymbol{0}$ and $\boldsymbol{e}$) and the integrality constraints, $\mathcal{F}_{\text{SPP}}$ contains only $\{0,1\}$-vectors, i.e., $\mathcal{F}_{\text{SPP}} \subseteq \{0,1\}^n$. Finally, the linear relaxation of SPP, denoted SPP^{LR}, is the linear program obtained by removing the integrality constraints. Its feasible domain is $\mathcal{F}_{\text{SPP}^{\text{LR}}}$ and its optimal value is $z^\star_{\text{SPP}^{\text{LR}}}$. If k augmentations have been performed, $\boldsymbol{x}^k$ designates a known integer solution that we seek to improve, and $\boldsymbol{x}^{k+1}$ is the next solution that we want to determine.

The SPP has specific features that allow the design of integral simplex methods. First, SPP is a $\{0,1\}$-program, so all the integer points of the domain of $\mathcal{F}_{\text{SPP}^{\text{LR}}}$ are extreme points of that polytope. Second, as shown by Trubin (1969), it possesses the *quasi-integral* property, also called the *Trubin property*: every edge of the convex hull of the integer domain $Conv\,(\mathcal{F}_{\text{SPP}})$ is also an edge of $\mathcal{F}_{\text{SPP}^{\text{LR}}}$. These two observations show that from any initial solution there exists a finite sequence of strictly augmenting integer solutions $\left(\boldsymbol{x}^k\right)_k$ that reaches an optimal solution and has the property that any two consecutive points are adjacent in $\mathcal{F}_{\text{SPP}^{\text{LR}}}$. Two consecutive points are neighbors in $\mathcal{F}_{\text{SPP}^{\text{LR}}}$, so it is possible to go from one to the other by performing a sequence of simplex pivots. Moreover, it is always possible to find a sequence of pivots between two adjacent vertices such that only one of them is nondegenerate; otherwise they would not be adjacent. An integral simplex method is thus a primal algorithm in which an augmentation is performed through one or several simplex pivots, only one of which is nondegenerate.

SPPs have intrinsic degeneracy, and this is a challenge for primal algorithms, especially algorithms based on simplex pivots. Therefore, integral simplex implementations need techniques that are specifically designed to cope with degeneracy. One of the first algorithms to take degeneracy into account was that of Balas & Padberg (1975). Given an initial solution $\boldsymbol{x}^k$ of SPP, they generate a large number of augmenting directions and consider only those that lead to integral neighbors of $\boldsymbol{x}^k$. For an integer neighbor $\boldsymbol{x}^{k+1}$ of $\boldsymbol{x}^k$, let $\boldsymbol{d} = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k$ be the corresponding direction. The positive support of $\boldsymbol{d}$, $\mathcal{Q}^+ = \{j | d_j > 0\}$, defines the

set of indices of columns that should be pivoted into the basis to go from $\boldsymbol{x}^k$ to $\boldsymbol{x}^{k+1}$. In the terminology of Balas & Padberg (1975), $\mathcal{Q}^+$ is *nondecomposable* if, for any strict subset $\tilde{\mathcal{Q}}^+ \subsetneq \mathcal{Q}^+$, performing pivots on the variables indexed by $\tilde{\mathcal{Q}}^+$ results in no change in the solution (degenerate pivots only). They prove that $\mathcal{Q}^+$ is nondecomposable if and only if $\boldsymbol{x}^{k+1}$ is a neighbor of $\boldsymbol{x}^k$. After restricting the set of directions to those that are integral, they choose the steepest and either perform the corresponding pivots if it is augmenting, or prove that $\boldsymbol{x}^k$ is optimal. If $\boldsymbol{x}^k$ is not optimal, $\boldsymbol{x}^{k+1}$ is different from $\boldsymbol{x}^k$ and has a strictly better cost. The main handicap of this method is that the number of integral neighbors of $\boldsymbol{x}^k$ and the number of potential directions that must be eliminated explode with the dimension of the search space (n).

Thompson (2002) introduces a *local integral simplex method* (Lism) and a *global integral simplex method* (Glism). Lism is based on a sequence of *pivots-on*-1, i.e., pivots on entries of the simplex tableau that take the value 1, and it is limited to augmenting pivots. For the SPP, given a basic solution $\boldsymbol{x}^k$, pivots-on-1 are exactly the simplex pivots that lead to a neighboring integer solution of $\boldsymbol{x}^k$. It may however be impossible to reach optimality by performing only augmenting pivots-on-1, and Thompson (2002) therefore embeds Lism into the Glism framework. In Glism, a branching tree is explored, and its subproblems are solved using Lism. At each node of the tree, if Lism solves the subproblem to optimality, the branch no longer needs to be explored; otherwise, new branches are created and the process continues until the tree has been exhaustively explored. Thompson (2002) shows promising numerical results on the Hoffman and Padberg benchmark (Hoffman & Padberg, 1993). Constraint propagation methods are used to speed up the solution of the linear relaxations, but the restriction to pivots-on-1 may lead to an explosion of the size of the branching tree. Although the total number of simplex pivots to reach optimality cannot exceed $2(n^2+n)^m$ (we do not doubt the theoretical importance of that result), this bound is of limited interest in practice. For example, in aircrew scheduling, m is typically the number of flights (often over $1,000$) and n the number of pairings (usually over $10,000$). Saxena (2003a,b) extends the work of Thompson, exploring the characterization of neighboring solutions and the corresponding bases, anti-cycling techniques, and cutting planes. His work is mainly theoretical.

Stallmann & Brglez (2007) present a comparative study of their augmentation algorithm for the set covering problem (a variation of the SPP in which the equality constraints are replaced by $\geq$ inequalities). Although they give no major theoretical improvements, their work is an excellent example of a comparison between an all-integer algorithm and a standard solver based on branch-and-bound (IBM CPLEX, in this case). It is one of the few studies that give numerical results for sizable instances (up to 2,900 constraints).

One of the most interesting recent studies of integral simplex algorithms is that carried out by Rönnberg and Larsson as part of Rönnberg's PhD (Rönnberg, 2012). They propose an all-integer column-generation scheme (Rönnberg & Larsson, 2009) that they apply to the nurse scheduling problem (Rönnberg & Larsson, 2014). Their algorithm allows both pivots-on-1 and pivots-on-(-1). The latter yield no change in the solution, so the approach is vulnerable to degeneracy.

Another pivot-based method in the same spirit is the ISUD of Zaghrouti et al. (2014). It is a promising recent development because it is based on linear programming techniques that take advantage of degeneracy and it guarantees strict improvement at each iteration (see Elhallaoui et al. (2011); Omer et al. (2015)). To find the edge leading to the next point, one solves a linear program to select an augmenting direction for the current point from the cone of feasible directions. To ensure that this linear program is bounded (the directions could go to infinity), a normalization constraint is added and the optimization is performed on a section of the cone. Solving this program with the simplex algorithm guarantees that the augmenting direction returned corresponds to an extreme ray of the cone and therefore to an edge of $\mathcal{F}_{\text{SPP}^{\text{LR}}}$. The solution of this linear program strongly depends on the chosen normalization weights, and so does the likelihood that the next solution is integer. Zaghrouti et al. (2014) report promising results for medium aircrew scheduling and large bus driver scheduling instances. They consider all the weights to be equal to 1 in the normalization constraint; they therefore call it the *convexity* constraint. This linear program usually produces integral augmentations; when it does not, branching is necessary. Rosat et al. (2015a) strengthen the theoretical framework of ISUD, introduce a generic normalization constraint, explore the theoretical properties of some specific constraints, and discuss the practical design of the normalization constraint. They report computational results that improve on those of the original implementation. They also compare different normalization strategies and highlight their influence on the behavior of the algorithm. The present work focuses on a dynamic update of the normalization constraint to penalize fractional directions and increase the likelihood that the edge leads to an integer neighbor of $\boldsymbol{x}^k$.

6.3 The integral simplex using decomposition

In this section, we present the ISUD algorithm as it is described by Rosat et al. (2015a) (i.e., in the case of a generic normalization constraint) and discuss some of the main theoretical results.

6.3.1 Notation

We first introduce some notation, similar to that of Rosat et al. (2015a). For any polyhedron P, $Ext\,(P)$ is the set of its extreme points. We use lower-case bold symbols for column vectors and upper-case bold symbols for matrices. Given subsets $\mathcal{I} \subseteq \{1,\ldots,m\}$ of the row indices and $\mathcal{J} \subseteq \{1,\ldots,n\}$ of the column indices, the submatrix of $\boldsymbol{A}$ with rows indexed by $\mathcal{I}$ and columns indexed by $\mathcal{J}$ is denoted $\boldsymbol{A}_{\mathcal{I}\mathcal{J}}$. Similarly, $\boldsymbol{A}_{\mathcal{I}\cdot}$ is the set of rows of $\boldsymbol{A}$ indexed by $\mathcal{I}$, $\boldsymbol{A}_{\cdot\mathcal{J}}$ is the set of columns of $\boldsymbol{A}$ indexed by $\mathcal{J}$, and for any vector $\boldsymbol{v} \in \mathbb{R}^n$, $\boldsymbol{v}_{\mathcal{J}}$ is the subvector of all v_j, $j \in \mathcal{J}$. For any set $\mathcal{X}$, $\mathbb{1}_{\mathcal{X}}$ is the indicator vector of $\mathcal{X}$ with size dictated by the context, i.e., $[\mathbb{1}_{\mathcal{X}}]_j = 1$ if $j \in \mathcal{X}$, 0 otherwise. Finally, given any matrix $\boldsymbol{M}$, $\boldsymbol{M}^T$ is its transpose, and $\mathsf{Span}\,(\boldsymbol{M})$ is the linear span of its columns, also called the image of $\boldsymbol{M}$. Recall that, assuming that k augmentations have been performed, $\boldsymbol{x}^k$ designates a known integer solution that we seek to improve, and $\boldsymbol{x}^{k+1}$ is the next solution that we want to determine. The sets of indices of the positive and zero-valued variables of $\boldsymbol{x}^k$ are respectively denoted $\mathcal{P}^k$ and $\mathcal{Z}^k$ ($\mathcal{P}$ and $\mathcal{Z}$ when the context is clear). Hence, $\boldsymbol{x}^k_{\mathcal{P}} = \boldsymbol{e}$ and $\boldsymbol{x}^k_{\mathcal{Z}} = \boldsymbol{0}$. The set $\{1,\ldots,n\}$ is thus partitioned into $(\mathcal{P},\mathcal{Z})$. Set $\mathcal{P}$ is called the *reduced basis* and has cardinality $p = |\mathcal{P}|$. Without loss of generality, the columns of $\boldsymbol{A}$ are assumed to be ordered such that $\mathcal{P} = \{1,\ldots,p\}$. In this paper, the terms basis, basic, and nonbasic refer to this reduced basis $\mathcal{P}$.

Remark. The main differences between $\mathcal{P}$ and a classical simplex basis are that p may be smaller than m, and an extreme solution of $\mathrm{SPP}^{\mathrm{LR}}$ is associated with a single reduced basis instead of potentially many simplex bases. When $p < m$, the set of columns of $\boldsymbol{A}_{\cdot\mathcal{P}}$ is not a basis of $\mathbb{R}^m$ in the algebraic sense but only a set of linearly independent vectors that spans a subspace of $\mathbb{R}^m$, $\mathsf{Span}\,(\boldsymbol{A}_{\cdot\mathcal{P}})$. In the classical simplex method, the reduced basis is completed with columns from $\boldsymbol{A}$, forming a basis of $\mathbb{R}^m$. Entering a column of $\boldsymbol{A}$ of negative reduced cost that is in $\mathsf{Span}\,(\boldsymbol{A}_{\cdot\mathcal{P}})$ into the basis produces a nondegenerate pivot that improves the solution. Such pivots are however not sufficient to reach optimality, and the columns of $\boldsymbol{A}$ that are not in $\mathsf{Span}\,(\boldsymbol{A}_{\cdot\mathcal{P}})$ must also be considered. To find a nondegenerate improvement with these columns, we can proceed as follows: we find a linear combination of these columns such that this combination (or *surrogate column*) is in $\mathsf{Span}\,(\boldsymbol{A}_{\cdot\mathcal{P}})$. If its reduced cost is negative, the solution is strictly improved by iteratively pivoting all the columns of the combination into the working basis. Further discussion of degeneracy and reduced bases can be found in the work of Omer et al. (2015).

6.3.2 Fractional augmentation

Given $\boldsymbol{x}^k \in \mathcal{F}_{\text{SPP}}$ and a direction $\boldsymbol{d} \in \mathbb{R}^n$, $\boldsymbol{d}$ is called a *feasible* direction at $\boldsymbol{x}^k$ if it is possible to take a nonzero step in this direction while remaining feasible for SPP^{LR}, i.e., if $\exists \rho > 0 | \left(\boldsymbol{x}^k + \rho \boldsymbol{d}\right) \in \mathcal{F}_{\text{SPP}^{\text{LR}}}$. Moreover, it is called an *augmenting* direction if it has a negative cost ($\boldsymbol{c}^T\boldsymbol{d} < 0$). The largest ρ such that $\left(\boldsymbol{x}^k + \rho \boldsymbol{d}\right)$ is in $\mathcal{F}_{\text{SPP}^{\text{LR}}}$ is called the *maximal feasible step* in direction $\boldsymbol{d}$ from $\boldsymbol{x}^k$ and denoted $r(\boldsymbol{d})$ (or r or r^k when the context is clear). The set of all feasible directions at $\boldsymbol{x}^k$ is the cone $\{\boldsymbol{d} \in \mathbb{R}^n | \boldsymbol{A}\boldsymbol{d} = \boldsymbol{0}, \boldsymbol{d}_{\mathcal{P}} \leq \boldsymbol{0}, \boldsymbol{d}_{\mathcal{Z}} \geq \boldsymbol{0}\}$. The conditions $\boldsymbol{d}_{\mathcal{P}} \leq \boldsymbol{0}$ and $\boldsymbol{d}_{\mathcal{Z}} \geq \boldsymbol{0}$ are equivalent because all the entries of $\boldsymbol{A}$ are nonnegative. Furthermore, whenever this cone is not the singleton $\{\boldsymbol{0}\}$ (it is never a singleton in practice, since then $\mathcal{F}_{\text{SPP}}$ would also be a singleton), it is unbounded. Since $\boldsymbol{d} = \boldsymbol{0}$ is not an interesting direction if we seek to move from $\boldsymbol{x}^k$ to another solution, instead of considering the cone of the feasible directions, we will consider only a section of that cone. Let $\boldsymbol{w} \in \mathbb{R}^n$ be a normalization vector, and let

$$\Delta_{\boldsymbol{w}} = \left\{\boldsymbol{d} \in \mathbb{R}^n | \boldsymbol{A}\boldsymbol{d} = \boldsymbol{0}, \boldsymbol{w}^T\boldsymbol{d} = 1, \boldsymbol{d}_{\mathcal{Z}} \geq \boldsymbol{0}\right\} \tag{6.1}$$

be a section of the cone. A geometric description of $\Delta_{\boldsymbol{w}}$ is given in Figure 6.1. To ensure that $\Delta_{\boldsymbol{w}}$ is a proper section of the cone, we restrict our choice to $\boldsymbol{w}_{\mathcal{P}} \leq \boldsymbol{0}$ and $\boldsymbol{w}_{\mathcal{Z}} > \boldsymbol{0}$. The fractional augmentation problem (i.e., the problem of finding an augmented solution of the linear relaxation SPP^{LR} or asserting that $\boldsymbol{x}^k$ is optimal for SPP^{LR}) is therefore equivalent to determining whether or not the optimal value of the program, called the *maximum normalized augmentation*,

$$z^{\star}_{\text{MNA}^{\boldsymbol{w}}} = \min\left\{\boldsymbol{c}^T\boldsymbol{d} \,|\, \boldsymbol{d} \in \Delta_{\boldsymbol{w}}\right\} \tag{MNA$^{\boldsymbol{w}}$}$$

is negative. If it is negative, then any solution with a negative objective value is an augmenting feasible direction and the current solution can be (at least fractionally) improved. If it is nonnegative, no feasible direction at $\boldsymbol{x}^k$ is augmenting and $\boldsymbol{x}^k$ is therefore optimal for SPP^{LR} (and SPP).

Remark. The higher the weight associated with a variable, the less interesting the directions involving this variable. Increasing w_j pushes d_j toward zero and therefore tends to prevent $\boldsymbol{A}_{\cdot j}$ from entering the working basis. This can be seen as follows: let $\boldsymbol{w}^1$ and $\boldsymbol{w}^2$ be normalization vectors such that $\boldsymbol{w}^2$ is equal to $\boldsymbol{w}^1$ except for a given $j \in \mathcal{Z}$ for which $w^2_j > w^1_j$. Given $\boldsymbol{d}$ in the cone of feasible directions, let $\boldsymbol{d}^1 = \boldsymbol{d}/(\boldsymbol{w}^1)^T\boldsymbol{d}$ and $\boldsymbol{d}^2 = \boldsymbol{d}/(\boldsymbol{w}^2)^T\boldsymbol{d}$ be the corresponding normalized directions in $\Delta_{\boldsymbol{w}^1}$ and $\Delta_{\boldsymbol{w}^2}$. If $d_j = 0$, the costs of these directions are equal, whereas if $d_j > 0$, $\boldsymbol{c}^T\boldsymbol{d}^1 < \boldsymbol{c}^T\boldsymbol{d}^2$. In the latter case, the normalized direction *appears* to be better if $w_1 < w_2$. Of course, the cost of a normalized direction is

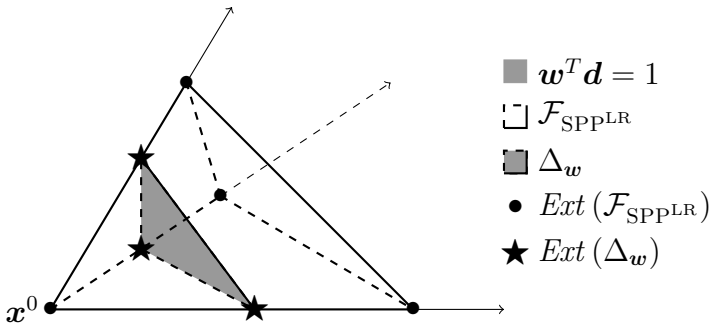


Figure 6.1 Geometric description of $\Delta_{\boldsymbol{w}}$. The cone of all feasible directions at $\boldsymbol{x}^0$ is defined by the three arrows. The grey area represents $\Delta_{\boldsymbol{w}}$, the set of normalized feasible directions.

seen through the prism of $\boldsymbol{w}$. The determination of the maximal *normalized* augmenting direction (MNA$^{\boldsymbol{w}}$) provides no information on the length of the maximal step, so a direction that *appears* to be better than another one may not be better in practice.

6.3.3 Integral augmentation

In the previous section, we have solved the problem of a fractional augmentation, i.e., an augmentation within the linear relaxation polytope. To tackle the integral augmentation problem **iAUG**, we introduce further notation. A feasible direction $\boldsymbol{d} \in \mathbb{R}^n$ is called an *integral* direction if it leads to another integer solution of SPP, i.e., $\exists \rho > 0 | \left(\boldsymbol{x}^k + \rho \boldsymbol{d}\right) \in \mathcal{F}_{\text{SPP}}$. Proposition 31 (Rosat et al., 2015a) gives a characterization of extreme integral directions.

Definition 10. A set of indices $\mathcal{S} \subseteq \{1, \ldots, n\}$ is *column-disjoint* if no pair of columns of $\boldsymbol{A}_{\cdot\mathcal{S}}$ has a common nonzero entry, i.e., $\forall i, j \in \mathcal{S}, i \neq j, \forall k \in \{1, \ldots, m\}, A_{ki}A_{kj} = 0$.

By extension, $\boldsymbol{A}_{\cdot\mathcal{S}}$ and $\boldsymbol{x}_{\mathcal{S}}$ are said to be column-disjoint if $\mathcal{S}$ is. The set of all integral directions is a subset of $\Delta_{\boldsymbol{w}}$ denoted $\Delta^{int}_{\boldsymbol{w}}$. Since $\boldsymbol{A}$ is binary, $\mathcal{S}$ is column-disjoint iff $\boldsymbol{A}_{\cdot\mathcal{S}}$ is a set of orthogonal columns.

Proposition 31. (Proposition 7 of Rosat et al. (2015a)) *Given $\boldsymbol{d}^\star \in Ext\,(\Delta_{\boldsymbol{w}})$, $\mathcal{S}^+ = Supp\,(\boldsymbol{d}^\star_{\mathcal{Z}})$, and $\mathcal{S}^- = Supp\,(\boldsymbol{d}^\star_{\mathcal{P}})$, ($\boldsymbol{d}^\star$ is an integral direction) $\Leftrightarrow$ ($\mathcal{S}^+$ is column-disjoint). For any such direction, $d^\star_j = -\delta_0$ if $j \in \mathcal{S}^-$, δ_0 if $j \in \mathcal{S}^+$, and 0 otherwise, where $\delta_0 = (\sum_{j\in\mathcal{S}^-\cup\mathcal{S}^+} |w_j|)^{-1} = (-\sum_{j\in\mathcal{S}^-} w_j + \sum_{j\in\mathcal{S}^+} w_j)^{-1}$. Moreover, the maximal step in that direction is $r = 1/\delta_0$.*

Because SPP is quasi-integral, the edges of $Conv\,(\mathcal{F}_{\text{SPP}})$ are edges of $\mathcal{F}_{\text{SPP}^{\text{LR}}}$, so $Ext\,(\Delta_{\boldsymbol{w}}^{int}) \subseteq Ext\,(\Delta_{\boldsymbol{w}})$. In practice, $\text{MNA}^{\boldsymbol{w}}$ is solved with the simplex algorithm, and the optimal solution belongs to $Ext\,(\Delta_{\boldsymbol{w}})$. By Proposition 31, it is sufficient to test whether the solution of $\text{MNA}^{\boldsymbol{w}}$ is column-disjoint to determine if it is integral. Furthermore, Rosat et al. (2015a) show that, given any extreme feasible direction $\bar{\boldsymbol{d}}$ of negative cost ($\boldsymbol{c}^T\boldsymbol{d} < 0$), there exists a weight vector $\boldsymbol{w}$ such that the optimal solution $\boldsymbol{d}^\star$ of $\text{MNA}^{\boldsymbol{w}}$ is positively proportional to $\bar{\boldsymbol{d}}$. More precisely, the normalized solution is $\boldsymbol{d}^\star = \bar{\boldsymbol{d}}/\boldsymbol{w}^T\bar{\boldsymbol{d}}$ and minimizes $\left(\boldsymbol{c}^T\boldsymbol{d}/\boldsymbol{w}^T\boldsymbol{d}\right)$ over the whole cone of feasible directions (and also over $\Delta_{\boldsymbol{w}}$). The solution of $\text{MNA}^{\boldsymbol{w}}$, i.e., the direction proposed by the algorithm, thus strongly depends on the chosen normalization weights, and so does the likelihood that the next solution is integer. Rosat et al. (2015a) also give advice on how to determine good normalization weights and offer four strategies that we summarize here. Note that when we write $\boldsymbol{w}$ as the concatenation of two vectors $(\boldsymbol{u}, \boldsymbol{v})$, we mean that $\boldsymbol{w}_{\mathcal{P}} = \boldsymbol{u}$ and $\boldsymbol{w}_{\mathcal{Z}} = \boldsymbol{v}$.

1. MMA: Nonweighted normalization vector $\boldsymbol{w} = (-\boldsymbol{e}, \boldsymbol{e})$. The corresponding problem $\text{MNA}^{(-\boldsymbol{e},\boldsymbol{e})}$ is called the *maximum mean augmentation* problem. Theoretical results on the maximum number of augmentations to reach optimality have been given for this normalization constraint (see Spille & Weismantel (2005)).
2. MIMA: Nonweighted normalization vector that applies weights only to the nonbasic (or incoming) variables $\boldsymbol{w} = (\boldsymbol{0}, \boldsymbol{e})$. The corresponding problem $\text{MNA}^{(\boldsymbol{0},\boldsymbol{e})}$ is called the *minimum incoming mean augmentation* problem. This weight vector appears in the seminal work of Zaghrouti et al. (2014). It has interesting theoretical properties, particularly when used in the decomposition context (see Rosat et al. (2015a)).
3. NORM: The weight w_j of every nonbasic variable d_j equals the number of nonzero entries of the corresponding column of $\boldsymbol{A}$, called the *norm* of the column and denoted n_j: $\boldsymbol{w} = (\boldsymbol{0}, \boldsymbol{n})$. The higher the weight, the less likely that the variable enters the basis. This weight vector therefore tends to prevent denser columns from entering the basis and thus fosters disjoint solutions.
4. DEG: The weight w_j of every nonbasic variable d_j equals its *incompatibility degree* w.r.t. the current working basis $\mathcal{P}$, denoted ι_j: $\boldsymbol{w} = (\boldsymbol{0}, \boldsymbol{\iota})$. The incompatibility degree is described in Section 6.3.4.

The reader may object that comparing the optimal values of $\text{MNA}^{\boldsymbol{w}}$ and $\text{MNA}^{\boldsymbol{w}'}$ for different normalization constraints does not make sense ($\text{MNA}^{\boldsymbol{w}}$ minimizes $\boldsymbol{c}^T\boldsymbol{d}/\boldsymbol{w}^T\boldsymbol{d}$ over the cone of feasible directions, while $\text{MNA}^{\boldsymbol{w}'}$ minimizes $\boldsymbol{c}^T\boldsymbol{d}/\boldsymbol{w}'^T\boldsymbol{d}$ over the same domain). For this reason, we define the mean scaled cost of a direction that will allow us to compare the

quality of two directions and, in particular, of the optimal solutions of MNA$^{\boldsymbol{w}}$ and MNA$^{\boldsymbol{w'}}$ when $\boldsymbol{w} \neq \boldsymbol{w'}$. This will be important later.

Definition 11. The *mean scaled cost* of direction $\boldsymbol{d}$ is the cost of its mean scaling $\boldsymbol{d}^e = \boldsymbol{d}/(-\boldsymbol{e}, \boldsymbol{e})^T\boldsymbol{d} = \boldsymbol{d}/(\sum_{j\in\{1,\dots,n\}} |d_j|)$ (corresponding to the maximum mean augmentation MMA mentioned above).

6.3.4 Incompatibility degree of a column

Let $\mathsf{V}_1 = \mathsf{Span}\,(\boldsymbol{A}_{\cdot\mathcal{P}})$ be the subspace of $\mathbb{R}^m$ spanned by $\boldsymbol{A}_{\cdot\mathcal{P}}$. By definition, the columns of $\boldsymbol{A}_{\cdot\mathcal{P}}$ are linearly independent, hence they form a basis $\boldsymbol{A}_{\cdot\mathcal{P}}$ of V_1. One can complete this basis with a set of $m-p$ independent vectors $\boldsymbol{B_2} = [\boldsymbol{v^{p+1}} \dots \boldsymbol{v^m}]$ and denote by V_2 the subspace spanned by these vectors. By definition, V_1 and V_2 are supplementary subspaces of $\mathbb{R}^m$ ($\mathbb{R}^m = \mathsf{V}_1 \oplus \mathsf{V}_2$). The matrix built by the concatenation of the aforementioned bases $\boldsymbol{B} = \begin{bmatrix}\boldsymbol{A}_{\cdot\mathcal{P}} & \boldsymbol{B_2}\end{bmatrix}$ is nonsingular by construction; its inverse is denoted $\boldsymbol{B}^{-1}$. One can freely transform the linear equations that define $\Delta_{\boldsymbol{w}}$ by multiplying them by $\boldsymbol{B}^{-1}$, so $\boldsymbol{Ad} = \boldsymbol{0}$ becomes $\boldsymbol{B}^{-1}\boldsymbol{Ad} = \boldsymbol{0}$. Because the current solution $\boldsymbol{x}^k$ is integral and because of the nature of the SPP, one can reorder the rows of $\boldsymbol{A}$ such that

$$\boldsymbol{A} = \begin{bmatrix}\boldsymbol{I}_p & \boldsymbol{A}_{\mathcal{P}\mathcal{Z}} \\ \boldsymbol{0} & \boldsymbol{A}_{\bar{\mathcal{P}}\mathcal{Z}}\end{bmatrix}, \text{ and therefore } \bar{\boldsymbol{A}} = \boldsymbol{B}^{-1}\boldsymbol{A} = \begin{bmatrix}\boldsymbol{I}_p & \bar{\boldsymbol{A}}_{\mathcal{P}\mathcal{Z}} \\ \boldsymbol{0} & \bar{\boldsymbol{A}}_{\bar{\mathcal{P}}\mathcal{Z}}\end{bmatrix}, \tag{6.2}$$

where $\mathcal{P} = \{1, \dots, p\}$ and $\bar{\mathcal{P}} = \{p+1, \dots, m\}$. For $j \in \{1, \dots, n\}$, the *incompatibility degree* ι_j of the column $\boldsymbol{A}_j$ w.r.t. $\mathcal{P}$ and $\boldsymbol{B}$ is the number of nonzero entries in the lower part of $\bar{\boldsymbol{A}}_{\cdot j}$, i.e., in $\bar{\boldsymbol{A}}_{\bar{\mathcal{P}}j}$. Column $\boldsymbol{A}_{\cdot j}$ is said to be ι_j-incompatible, and 0-incompatible columns are called *compatible* columns. As can be seen in Equation (6.2), all columns from the working basis $\boldsymbol{A}_{\cdot P}$ are compatible. In practice, V_2 and $\boldsymbol{B_2}$ are chosen so that $\boldsymbol{B}^{-1}$ is easy to compute. Basis completion is discussed further by Bouarab et al. (2014), and the reader is encouraged to read the work of Gauthier et al. (2015) for a global overview of the underlying vector space decomposition structure. To the best of our knowledge, however, this is the first time that Proposition 32 below has been stated in these terms. We think that this result provides interesting insight and clarifies the notion of incompatibility degree. The reader should note that the vector spaces V_1 and V_2 may not be orthogonal. For the proposition, let $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$ respectively be the projection on V_1 parallel to V_2 and on V_2 parallel to V_1, i.e., for any vector $\boldsymbol{v} \in \mathbb{R}^m$, $\boldsymbol{v} = \boldsymbol{\pi}_1(\boldsymbol{v}) + \boldsymbol{\pi}_2(\boldsymbol{v})$, $\boldsymbol{\pi}_1(\boldsymbol{v}) \in \mathsf{V}_1$ and $\boldsymbol{\pi}_2(\boldsymbol{v}) \in \mathsf{V}_2$.

Proposition 32. *With the above definitions, the incompatibility degree of column $\boldsymbol{A}_{\cdot j}$ is the number of nonzero coefficients in the expression of $\boldsymbol{\pi}_2(\boldsymbol{v})$ as a linear combination of*

elements of the basis $\boldsymbol{B_2}$ *of* V_2*. It is equal to the number of elements of* $\boldsymbol{B_2}$ *involved in the decomposition of* $\boldsymbol{v}$ *on basis* $\boldsymbol{B} = \begin{bmatrix}\boldsymbol{A}_{\cdot\mathcal{P}} & \boldsymbol{B_2}\end{bmatrix}$.

Proof. Let $j \in \{1, \dots, n\}$. Because $\boldsymbol{B}$ is a basis of $\mathbb{R}^m$, there exists a single $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that $\boldsymbol{A}_{\cdot j} = \sum_{i=1}^{p} \lambda_i \boldsymbol{A}_{\cdot i} + \sum_{i=p+1}^{m} \lambda_i \boldsymbol{v}_i$. By definition of $\boldsymbol{B}^{-1}$ and $\bar{\boldsymbol{A}}$,

$$\bar{\boldsymbol{A}}_{\cdot j} = \boldsymbol{B}^{-1}\boldsymbol{A}_{\cdot j} = \left(\sum_{i=1}^{p} \lambda_i \boldsymbol{B}^{-1}\boldsymbol{A}_{\cdot i} + \sum_{i=p+1}^{m} \lambda_i \boldsymbol{B}^{-1}\boldsymbol{v^i}\right) = \sum_{i=1}^{m} \lambda_i \boldsymbol{\epsilon}^i, \tag{6.3}$$

where for all $i \in \{1, \dots, m\}$, $\boldsymbol{\epsilon}^i$ is the vector defined by $\epsilon^i_j = 1$ if $i = j$ and 0 otherwise. Thus, ι_j is the number of nonzero elements in $\boldsymbol{\lambda}_{\bar{\mathcal{P}}}$. Since $\boldsymbol{\pi}_1(\boldsymbol{v}) = \sum_{i=1}^{p} \lambda_i \boldsymbol{A}_{\cdot i}$ and $\boldsymbol{\pi}_2(\boldsymbol{v}) = \sum_{i=p+1}^{m} \lambda_i \boldsymbol{v^i}$, the proposition holds. □

Proposition 32 shows the paramount importance of the choice of the vectors $\boldsymbol{v}^{p+1}, \dots, \boldsymbol{v}^m$ when completing the working basis $\boldsymbol{A}_{\cdot\mathcal{P}}$. For example, when we build a simplex basis, the choice is limited to columns of $\boldsymbol{A}_{\cdot\mathcal{Z}}$. Here, we extend that choice to any basis completion of $\mathbb{R}^m$. However, because $\boldsymbol{v^{p+1}}, \dots, \boldsymbol{v^m}$ are chosen arbitrarily, there is no point in performing what could be called degenerate pivots. These would result in a change of the completion but no change in the working basis $\boldsymbol{A}_{\cdot\mathcal{P}}$. A change of the completion is exactly what we do not want to perform, since it entails the computation of the new inverse matrix $\boldsymbol{B}^{-1}$ (whereas the chosen completion should ease this computation) and changes all the incompatibility degrees. We make the following recommendations regarding the choice of the basis completion. First, it should make the computation of $\boldsymbol{B}^{-1}$ fast; if the matrix inversion is slow it will jeopardize the whole process. Second, the columns of $\boldsymbol{A}$ that are of interest in the pursuit of optimality as well as those that are likely to appear in disjoint combinations should be as sparse as possible in $\bar{\boldsymbol{A}}$. If these conditions hold, using incompatibility degrees as normalization weights (the DEG constraint in Section 6.3.3) and/or performing partial pricing on low-ι columns give the algorithm a twofold advantage. Such a partial pricing is in the spirit of the multi-phase strategy that Elhallaoui et al. (2010) propose for the dynamic constraint aggregation algorithm. Our choice of basis completion yields incompatibility degrees as described by Rosat et al. (2015b).

6.3.5 Pseudocode

The practical implementation of ISUD borrows some ideas from variable neighborhood search to speed up the algorithm (see Algorithm 7). The set of columns considered in the augmentation problem is not $\mathcal{Z}$, but the restriction $\mathcal{Z}_\iota$ of $\mathcal{Z}$ to at most ι-incompatible variables, i.e., variables that are ι'-incompatible for $\iota' \leq \iota$. Here ι is called the *phase number*.

The corresponding restriction of $\text{MNA}^{\boldsymbol{w}}$ to variables in $\mathcal{P}$ and $\mathcal{Z}_\iota$ is denoted $\text{MNA}^{\boldsymbol{w}}_{\mathcal{Z}_\iota}$. We thus solve an augmentation problem in which at most ι-incompatible columns are allowed to enter the working basis. The algorithm stops either when the neighborhood comprises all the variables in $\mathcal{Z}$ or when a maximum phase number is reached (heuristic stopping variant, line 2 of Algorithm 7). For the sake of clarity, hereinafter we will consider solving the version of $\text{MNA}^{\boldsymbol{w}}$ in which all columns of $\mathcal{Z}$ are present. All the results can then be adapted by simply replacing $\mathcal{Z}$ by $\mathcal{Z}_\iota$. To increase the chance of finding integral augmenting directions, we implement a simple nonexhaustive branching (fixing) strategy. Whenever a non-column-disjoint augmenting direction $\boldsymbol{d}^\star$ is found, all the entering variables from $\boldsymbol{d}_\mathcal{Z}$ are set to zero and the augmentation problem is solved again. Variables fixed to zero are freed only when the phase number ι is increased.

Algorithm 7: Integral Simplex Using Decomposition

Input: $\boldsymbol{x}^0$, a solution of SPP; INC_MAX, maximum phase number.
Output: $\boldsymbol{x}^k$, a possibly better solution of SPP.

1 Compute $\mathcal{P}$ and $\mathcal{Z}_\iota$ associated with $\boldsymbol{x}^0$; $k := 0$; $\iota := 1$;
2 **while** $\iota < INC\_MAX$ *and* $\mathcal{Z}_{\iota-1} \neq \mathcal{Z}$ **do**
3 Solve $\text{MNA}^{\boldsymbol{w}}_{\mathcal{Z}_\iota}$;
4 **if** $\text{MNA}^{\boldsymbol{w}}_{\mathcal{Z}_\iota}$ *is infeasible* **then**
5 Remove all fixing constraints from $\text{MNA}^{\boldsymbol{w}}_{\mathcal{Z}_\iota}$; $\iota := \iota + 1$; Recompute $\mathcal{Z}_\iota$;
6 **else**
7 $\boldsymbol{d}^\star :=$ an (extreme) optimal solution of $\text{MNA}^{\boldsymbol{w}}_{\mathcal{Z}_\iota}$;
8 **if** $\boldsymbol{c}^T\boldsymbol{d}^\star < 0$ **then**
9 **if** $\boldsymbol{d}^\star_{\mathcal{Z}_\iota}$ *is column-disjoint* **then**
10 $r^\star :=$ maximal step in direction $\boldsymbol{d}^\star$; $\boldsymbol{x}^{k+1} := \boldsymbol{x}^k + r^\star\boldsymbol{d}^\star$; $k := k + 1$;
 Recompute $\mathcal{P}$ and $\mathcal{Z}_\iota$ according to the new solution;
11 **else**
12 For each $j \in \mathit{Supp}\,(\boldsymbol{d}^\star_\mathcal{Z})$, add the fixing constraint $d_j = 0$ to $\text{MNA}^{\boldsymbol{w}}_{\mathcal{Z}_\iota}$;
13 **else**
14 Remove all fixing constraints from $\text{MNA}^{\boldsymbol{w}}_{\mathcal{Z}_\iota}$; $\iota := \iota + 1$; Recompute $\mathcal{Z}_\iota$;

6.3.6 Further remarks on algorithm

Remark. Zaghrouti et al. (2014) show that $\Delta_{\boldsymbol{w}}$ can be projected into the space of $\boldsymbol{d}_\mathcal{Z}$ without loss of information. In that subspace, we need p fewer constraints and p fewer variables to describe the cone section, and we obtain the new linear constraints $\bar{\boldsymbol{A}}\boldsymbol{d}_\mathcal{Z} = 0$, costs $\bar{\boldsymbol{c}}$, and normalization weights $\bar{\boldsymbol{w}}$ from the original ones via an easy-to-determine linear transformation. This speeds up the solution process, and we therefore use this transformation in our implementation.

Remark. Rosat et al. (2015a) prove that the maximum normalized augmentation problem $\text{MNA}^{\boldsymbol{w}}$ can be decomposed into two subproblems that can be solved separately. Set $\mathcal{Z}$ is partitioned into two smaller subsets $\mathcal{C}$ and $\mathcal{I}$, where the indices in $\mathcal{C}$ are those of all the non-basic variables in $\mathsf{Span}\,(\boldsymbol{A}_{\cdot\mathcal{P}})$, i.e., the compatible variables, and $\mathcal{I}$ contains the others. Let $\text{MNA}^{\boldsymbol{w}}(\mathcal{C})$ and $\text{MNA}^{\boldsymbol{w}}(\mathcal{I})$ be the problems obtained from $\text{MNA}^{\boldsymbol{w}}$ by replacing $\mathcal{Z}$ with $\mathcal{C}$ and $\mathcal{I}$. Rosat et al. (2015a) prove that $z^{\star}_{\text{MNA}^{\boldsymbol{w}}} = \min\left\{z^{\star}_{\text{MNA}^{\boldsymbol{w}}(\mathcal{C})}, z^{\star}_{\text{MNA}^{\boldsymbol{w}}(\mathcal{I})}\right\}$. Solving $\text{MNA}^{\boldsymbol{w}}$ is therefore equivalent to solving these two problems independently. We also use this decomposition in our implementation.

For the sake of clarity, although we make use of the above two remarks, we describe all manipulations related to the augmentation using $\text{MNA}^{\boldsymbol{w}}$. We can transpose them all to the projection of $\Delta_{\boldsymbol{w}}$ by applying the same linear transformation that yields that projection, and we can adapt them to the decomposition by simply applying them to either subproblem.

6.4 Dynamic normalization in ISUD

This section presents the core of our algorithmic improvements to ISUD. In our new version of the algorithm, when the (extreme) optimal solution $\boldsymbol{d}^{\star}$ of $\text{MNA}^{\boldsymbol{w}}$ is fractional, we add a perturbation $\tilde{\boldsymbol{w}}$ to the weights to obtain a column-disjoint solution for the modified problem $\text{MNA}^{(\boldsymbol{w}+\tilde{\boldsymbol{w}})}$. In Section 6.4.1, we explain the design of that method and we give the pseudocode. In Section 6.4.2, we give geometric insight into and theoretical properties of the normalization vectors and their connection to the set of feasible directions. In Section 6.4.3, we detail our four update strategies, which will be compared in Section 6.6.

6.4.1 Design and algorithm

Suppose that the optimal solution $\boldsymbol{d}^{\star} \in \mathit{Ext}\,(\Delta_{\boldsymbol{w}})$ of $\text{MNA}^{\boldsymbol{w}}$ is a fractional direction of negative cost and that $\boldsymbol{x}^k$ is **not** optimal for SPP. It is important to recall here that all extreme integral directions of edges of the SPP linear relaxation are quasi-integral. Therefore, $\Delta_{\boldsymbol{w}}$ is a relaxation of $\Delta^{int}_{\boldsymbol{w}}$ with the property that $\mathit{Ext}\,(\Delta^{int}_{\boldsymbol{w}}) \subseteq \mathit{Ext}\,(\Delta_{\boldsymbol{w}})$. We aim to determine $\boldsymbol{w'}$ such that the optimal solution $\boldsymbol{d'}$ of $\text{MNA}^{\boldsymbol{w'}}$ is column-disjoint. From Rosat et al. (2015a), we know that such a $\boldsymbol{w'}$ exists. We propose a modification of the program ($\text{MNA}^{\boldsymbol{w}}$) that models the relaxation of **iAUG**. Another approach would be to tighten the relaxation polytope by adding cutting planes (see Rosat et al. (2015b)). We chose to act on the normalization constraint only because (1) we know that it is sufficient, (2) it is faster than adding cutting planes, and (3) adding cuts tends to increase the number of fractional solutions if the cuts are not facets. A modification of the normalization constraint does not change the nature of

the polytope; it simply scales the directions in a different way.

Our method results in the addition of lines 12–15 in the new version of ISUD (Algorithm 8). Regardless of the strategy chosen to compute the perturbation $\tilde{\boldsymbol{w}}$, PERT () is the function that computes this perturbation, and q is the number of consecutive updates that have already been performed. We may not need all these inputs, but in general the perturbation depends on the current solution $\boldsymbol{x}^k$, the (non-column-disjoint) current augmenting direction $\boldsymbol{d}^\star$, the weight vector $\boldsymbol{w}$, and q, so $\tilde{\boldsymbol{w}} = \texttt{PERT}\left(\boldsymbol{x}^k, \boldsymbol{d}^\star, \boldsymbol{w}, q\right)$ (line 13 of Algorithm 8). To ensure the termination of the process, we specify a maximum number Q of consecutive updates. Every time variables are fixed or the phase number is increased the counter q is reset to zero (lines 5 and 15).

Algorithm 8: ISUD with dynamic updates of the normalization weights

Input: $\boldsymbol{x}^0$, a solution of SPP; INC_MAX, maximum phase number considered; Q, maximum number of consecutive updates of normalization weights.
Output: $\boldsymbol{x}^k$, a possibly better solution of SPP.

```
1  Compute P and Z_ι associated with x^0; k := 0; ι := 1; q := 0;
2  while ι < INC_MAX and Z_{ι-1} ≠ Z do
3      Solve MNA^w_{|Z_ι};
4      if MNA^w_{|Z_ι} is infeasible then
5          Remove all fixing constraints from MNA^w_{|Z_ι}; ι := ι + 1; Recompute Z_ι; q := 0;
6      else
7          d* := an (extreme) optimal solution of MNA^w_{|Z_ι};
8          if c^T d* < 0 then
9              if d*_{Z_ι} is column-disjoint then
10                 r* := maximal step in direction d*; x^{k+1} := x^k + r* d*; k := k + 1;
                   Recompute P and Z_ι according to the new solution; q := 0;
11             else
12                 if q < Q then
13                     w̃ := PERT(x^k, d*, w, q); w := w + w̃; q := q + 1;
14                 else
15                     For each j ∈ Supp(d*_Z), add the fixing constraint d_j = 0 to MNA^w_{|Z_ι};
                       q := 0;
16         else
17             Remove all fixing constraints from MNA^w_{|Z_ι}; ι := ι + 1; q := 0; Recompute
               Z_ι;
```

6.4.2 Geometry of normalization weight vectors

We now give some geometric insights into the structure of the set of normalization constraints. This will provide a better understanding of our methods and their mechanics. We associate the weight vectors with optimal solutions of $\text{MNA}^{\boldsymbol{w}}$ and show that the set of

normalization constraints that yields the same optimal solution is a polytope. Let $\mathcal{W} = \left\{\boldsymbol{w} \in \mathbb{R}^n | \boldsymbol{w}_{\mathcal{P}} \leq \boldsymbol{0}, \boldsymbol{w}_{\mathcal{Z}} > \boldsymbol{0}, -\boldsymbol{e}^T\boldsymbol{w}_{\mathcal{P}} + \boldsymbol{e}^T\boldsymbol{w}_{\mathcal{Z}} = 1\right\}$ be the set of scaled normalization weight vectors that we wish to consider. As before, we assume that $\boldsymbol{w}_{\mathcal{P}} \leq \boldsymbol{0}$ and $\boldsymbol{w}_{\mathcal{Z}} > \boldsymbol{0}$ because these are sufficent conditions for Δ_w to be a proper section of the cone of feasible directions, without being too restrictive on the choice of $\boldsymbol{w}$. Let $\bar{\boldsymbol{w}} = \boldsymbol{w}/\left(-\boldsymbol{e}^T\boldsymbol{w}_{\mathcal{P}} + \boldsymbol{e}^T\boldsymbol{w}_{\mathcal{Z}}\right)$. Since $\text{MNA}^{\bar{\boldsymbol{w}}}$ is equivalent to $\text{MNA}^{\boldsymbol{w}}$, we can reasonably reduce the set of all possible normalization weights to $\mathcal{W}$. For every extreme direction $\boldsymbol{d}^e$ of the cone of feasible directions, let $\mathcal{W}(\boldsymbol{d}^e) = \left\{\boldsymbol{w} \in \mathcal{W} | \left(\boldsymbol{d}^e/\boldsymbol{w}^T\boldsymbol{d}^e\right) \in \arg\min \text{MNA}^{\boldsymbol{w}}\right\}$ be the set of normalization vectors $\boldsymbol{w} \in \mathcal{W}$ such that $\left(\boldsymbol{d}^e/\boldsymbol{w}^T\boldsymbol{d}^e\right)$ is an optimal solution of $\text{MNA}^{\boldsymbol{w}}$. In the same spirit, $\overset{\circ}{\mathcal{W}}(\boldsymbol{d}^e) = \left\{\boldsymbol{w} \in \mathcal{W} | \arg\min \text{MNA}^{\boldsymbol{w}} \text{ is the singleton } \left\{\boldsymbol{d}^e/\boldsymbol{w}^T\boldsymbol{d}^e\right\}\right\}$ is the set of normalization vectors $\boldsymbol{w} \in \mathcal{W}$ such that $\left(\boldsymbol{d}^e/\boldsymbol{w}^T\boldsymbol{d}^e\right)$ is the **only** optimal solution of $\text{MNA}^{\boldsymbol{w}}$. Obviously, $\overset{\circ}{\mathcal{W}}(\boldsymbol{d}^e) \subseteq \mathcal{W}(\boldsymbol{d}^e)$. In Proposition 34, we prove that $\overset{\circ}{\mathcal{W}}(\boldsymbol{d}^e)$ is the topological interior of $\mathcal{W}(\boldsymbol{d}^e)$, hence the chosen notation.

Proposition 33. *Suppose that $\boldsymbol{x}^k \in \mathcal{F}_{\text{SPP}}$ is not optimal for SPP^{LR}. Given an extreme direction $\boldsymbol{d}^e$, the following statements are equivalent:*

(i) $\boldsymbol{c}^T\boldsymbol{d}^e < 0$;

(ii) $\overset{\circ}{\mathcal{W}}(\boldsymbol{d}^e) \neq \emptyset$;

(iii) $\mathcal{W}(\boldsymbol{d}^e) \neq \emptyset$.

Proof. We prove that (i) $\Rightarrow$ (ii), (ii) $\Rightarrow$ (iii), and (iii) $\Rightarrow$ (i).

(i) $\Rightarrow$ (ii): This is proved in Lemma 11 of Rosat et al. (2015a).

(ii) $\Rightarrow$ (iii): $\overset{\circ}{\mathcal{W}}(\boldsymbol{d}^e) \subseteq \mathcal{W}(\boldsymbol{d}^e)$, so (ii) $\Rightarrow$ (iii).

(iii) $\Rightarrow$ (i): Suppose that $\mathcal{W}(\boldsymbol{d}^e) \neq \emptyset$ and let $\boldsymbol{w} \in \mathcal{W}(\boldsymbol{d}^e)$. Since $\boldsymbol{x}^k$ is nonoptimal for SPP^{LR}, there exists a normalized augmenting feasible direction $\boldsymbol{d}^\star \in \Delta_w$ at $\boldsymbol{x}^0$. By definition of $\mathcal{W}(\boldsymbol{d}^e)$, $\boldsymbol{d}^e/\left(\boldsymbol{w}^T\boldsymbol{d}^e\right)$ is optimal for $\text{MNA}^{\boldsymbol{w}}$, hence $\boldsymbol{c}^T\boldsymbol{d}^e/\left(\boldsymbol{w}^T\boldsymbol{d}^e\right) \leq \boldsymbol{c}^T\boldsymbol{d}^\star < 0$. Because $\boldsymbol{w}^T\boldsymbol{d}^e > 0$, the result holds.

□

Proposition 34. *Given an extreme direction $\boldsymbol{d}^e$, $\mathcal{W}(\boldsymbol{d}^e)$ is a polytope and $\overset{\circ}{\mathcal{W}}(\boldsymbol{d}^e)$ is its topological interior within $\mathcal{W}$.*

Proof. Let $\hat{\boldsymbol{w}} \in \mathcal{W}$ be a reference normalization vector. Since extreme directions of the cone

of feasible directions correspond to extreme points of any of its sections, we have

$$\begin{aligned}\mathcal{W}(\boldsymbol{d}^e) &= \left\{\boldsymbol{w} \in \mathcal{W} | \frac{\boldsymbol{d}^e}{\boldsymbol{w}^T\boldsymbol{d}^e} \in \arg\min \textsc{Mna}^{\boldsymbol{w}}\right\} \\ &= \left\{\boldsymbol{w} \in \mathcal{W} | \forall \boldsymbol{u} \in \mathit{Ext}(\Delta_{\hat{\boldsymbol{w}}}) \setminus \left\{\boldsymbol{d}^e / \hat{\boldsymbol{w}}^T\boldsymbol{d}^e\right\}, \frac{\boldsymbol{c}^T\boldsymbol{d}^e}{\boldsymbol{w}^T\boldsymbol{d}^e} \leq \frac{\boldsymbol{c}^T\boldsymbol{u}}{\boldsymbol{w}^T\boldsymbol{u}}\right\} \\ &= \left\{\boldsymbol{w} \in \mathcal{W} | \forall \boldsymbol{u} \in \mathit{Ext}(\Delta_{\hat{\boldsymbol{w}}}) \setminus \left\{\boldsymbol{d}^e / \hat{\boldsymbol{w}}^T\boldsymbol{d}^e\right\}, \left(\left(\boldsymbol{c}^T\boldsymbol{d}^e\right)\boldsymbol{u} - \left(\boldsymbol{c}^T\boldsymbol{u}\right)\boldsymbol{d}^e\right)^T \boldsymbol{w} \leq 0\right\}.\end{aligned}$$

Therefore, $\mathcal{W}(\boldsymbol{d}^e)$ is the intersection of $\mathcal{W}$ with a set P of semi-spaces (linear inequalities), which is a polyhedron. Because $\mathcal{W}(\boldsymbol{d}^e) \subseteq \mathcal{W}$ and $\mathcal{W}$ is a polytope, $\mathcal{W}(\boldsymbol{d}^e)$ is a polytope. The same reasoning can be applied to show that

$$\mathring{\mathcal{W}}(\boldsymbol{d}^e) = \left\{\boldsymbol{w} \in \mathcal{W} | \forall \boldsymbol{u} \in \mathit{Ext}(\Delta_{\hat{\boldsymbol{w}}}) \setminus \left\{\boldsymbol{d}^e / \hat{\boldsymbol{w}}^T\boldsymbol{d}^e\right\}, \left(\left(\boldsymbol{c}^T\boldsymbol{d}^e\right)\boldsymbol{u} - \left(\boldsymbol{c}^T\boldsymbol{u}\right)\boldsymbol{d}^e\right)^T \boldsymbol{w} < 0\right\}.$$

With that description, we see that $\mathring{\mathcal{W}}(\boldsymbol{d}^e)$ is the intersection of $\mathcal{W}$ with the interior of the same semi-spaces that define $\mathcal{W}(\boldsymbol{d}^e)$. Therefore, $\mathring{\mathcal{W}}(\boldsymbol{d}^e)$ is the topological interior of $\mathcal{W}(\boldsymbol{d}^e)$ within $\mathcal{W}$. □

Proposition 34 yields the geometric interpretation given in Figure 6.2. Our goal is to find $\boldsymbol{w}$ within one of the polytopes $\mathcal{W}(\boldsymbol{d})$ such that $\boldsymbol{d}$ is an integral direction. Experience tells us that in SPPs, the direction is likely to be integral. When it is not, we propose to modify $\boldsymbol{w}$ by adding an update $\tilde{\boldsymbol{w}}$ as shown in Figure 6.2 so as to find a point in another of the subpolytopes. Reaching a polytope whose interior is nonempty is in principle easier than converging toward a single point; the process is less sensitive to approximation errors. Two of our four strategies build the constraint anew and are akin to sampling methods. The others make smaller changes and aim to reach an adjacent polytope that is likely to be that of an integral direction. The idea behind these two methods is that the more the normalization weights are changed, the more we risk deteriorating the mean-scaled cost of the direction found.

6.4.3 Update strategies

We now summarize our four strategies for updating $\boldsymbol{w}$. The first attempt is always made with the incompatibility degrees as the normalization weights (Deg, $w_j = \iota_j$ if $j \in \mathcal{Z}$, 0 otherwise) because the four static versions of the algorithm compared by Rosat et al. (2015a) (listed in Section 6.3.3) perform well with this approach.

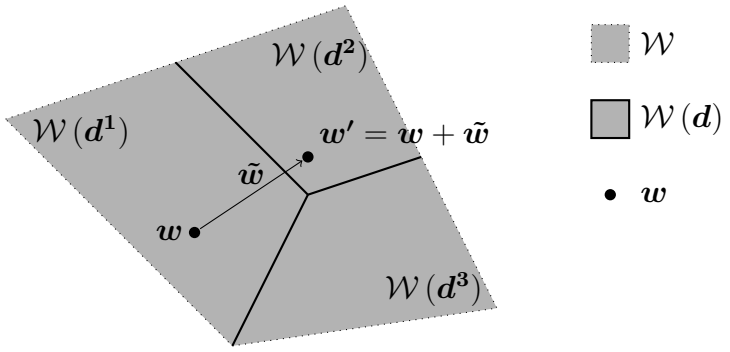


Figure 6.2 Geometric description of $\mathcal{W}$. The set of all available normalization constraints is the grey domain. Subpolytopes that correspond to $\mathcal{W}\left(\boldsymbol{d^i}\right)$ are delimited by thick black lines ($i \in \{1, 2, 3\}$). In this example, the only optimal solution of MNA$^{\boldsymbol{w}}$ is $\boldsymbol{d^1}$. When $\boldsymbol{w}$ is perturbed, $\boldsymbol{w'} := \boldsymbol{w} + \tilde{\boldsymbol{w}}$, the optimal solution changes because $\boldsymbol{w'} \notin \mathcal{W}(\boldsymbol{d^1})$, and the only optimal solution of MNA$^{\boldsymbol{w'}}$ is $\boldsymbol{d^2}$.

Random update (RAND)

The first modification is the simplest: random perturbation. For every index $j \in \mathcal{Z}$, the penalty $\tilde{w}_j$ is the value of a uniform integer random variable between 0 and θ, where θ is a parameter. All random variables are independent. This strategy is based on empirical observations that suggest that the likelihood of an integral optimal direction is relatively high regardless of the constraint. It also serves as a base for the comparison of the other methods. It can be interpreted as random sampling of the points in $\mathcal{W}$ at each iteration.

Alternate update (ALT)

This update applies each of the four constraints listed in Section 6.3.3. We consider them in the order of their respective performance in the static version of the algorithm (as reported by Rosat et al. (2015a)): DEG, NORM, MIMA, and MMA. The maximum number of consecutive updates Q is clearly four. This approach can be interpreted as the sampling of the same four points in $\mathcal{W}$ at each iteration.

Penalization of current fractional direction (DIR)

In the DIR update strategy, we perturb $\boldsymbol{w}$ so as to directly penalize the variables involved in the fractional direction $\boldsymbol{d}^\star$ returned by MNA$^{\boldsymbol{w}}$. For every index j in the support $\mathcal{S}^+$ of $\boldsymbol{d}^\star{}_{\mathcal{Z}}$, $\tilde{w}_j$ is increased by $\theta > 0$. Hence, $\tilde{\boldsymbol{w}} = \theta \mathbb{1}_{\mathcal{S}^+}$. Because the changeover from $\boldsymbol{w}$ to $\boldsymbol{w}' := \boldsymbol{w} + \tilde{\boldsymbol{w}}$ does not guarantee that the best augmenting direction $\boldsymbol{d}^\star$ changes, the value of θ is updated during the process depending on the success of previous attempts. On the one hand, the higher θ, the higher the probability that $\boldsymbol{d}^\star$ (or its scaling) is no longer optimal

for MNA$^{w'}$. On the other hand, the lower θ, the closer the solution of MNA$^{w'}$ to $\boldsymbol{d}^\star$ and the less deterioration in terms of the optimal value and quality of the augmentation. We handle this tradeoff in the following way: θ is given an initial value θ_0 (we report results for different values of θ_0). When we update the normalization weights, the new value of θ is

$$\theta := \begin{cases} 2\theta & \text{if the solution did not change } (\boldsymbol{d}^\star \text{ and } \boldsymbol{d'} \text{ are proportional}), \\ \theta/2 & \text{if the new solution } \boldsymbol{d'} \text{ is an integral direction}, \\ \theta/\lambda & \text{if } \boldsymbol{d'} \neq \boldsymbol{d}^\star \text{ and the mean scaled cost was deteriorated by a factor greater than 2}, \\ \theta & \text{otherwise}, \end{cases} \tag{6.4}$$

where $\boldsymbol{d}^\star$ and $\boldsymbol{d'}$ are optimal solutions of MNA^{w} and MNA$^{w'}$ respectively, and λ is a uniform random variable between 1 and 2. Three of these updates of θ are obviously consistent with the previous remarks, but the θ/λ update needs further explanation. When the solution changes but the cost is noticeably worse, the penalty is high enough to force the algorithm to seek out what can be seen as "bad" solutions; θ should therefore decrease. However, it is not desirable to restrict the possible values taken by θ to a discrete set. Our goal is instead to iteratively tune θ so that it reaches a tradeoff value, neither too large nor too small.

Remark. The penalization that we apply is the same for all variables of $\mathcal{S}^+$; it is not, for example, proportional to $\boldsymbol{d}^\star$. This is because we wish to foster column-disjoint solutions. A larger value for $d^\star_{j_1}$ than $d^\star_{j_2}$ in the fractional direction $\boldsymbol{d}^\star$ does not indicate that j_2 is more likely than j_1 to be part of a disjoint solution. Arguments can be found to justify both this position and the opposite. The propensity of a variable for appearing in disjoint or nondisjoint combinations is instead related to the nonzero entries of the corresponding column $\boldsymbol{A}_{\cdot j}$ and is taken into account because the base constraint is DEG. For this reason, we equally penalize each variable appearing in the nondisjoint direction $\boldsymbol{d}^\star$.

Update based on cutting planes (HYP)

In the HYP update strategy, we perturb $\boldsymbol{w}$ using the coefficient of the cutting planes that separate $\boldsymbol{d}^\star$ from the set of feasible integer directions $\Delta^{int}_{\boldsymbol{w}}$. We again assume that the optimal solution $\boldsymbol{d}^\star \in \mathit{Ext}\,(\Delta_{\boldsymbol{w}})$ of MNA^{w} found is not column-disjoint and of negative cost; we also assume that $\boldsymbol{x}^k$ is **not** optimal for SPP. Rosat et al. (2015b) provide separation algorithms for primal versions of odd-cycle and clique cuts that can be used in ISUD to separate such a direction from $\Delta^{int}_{\boldsymbol{w}}$. They prove that these cuts are of the form $\boldsymbol{\alpha}^T\boldsymbol{d} \leq 0$. We assume that a nonempty set $\mathcal{K}$ ($\mathcal{K}$ denotes both the set of cuts and that of their coefficient vectors since the context is clear enough) of such cutting planes has been generated, i.e., for every $\boldsymbol{\alpha} \in \mathcal{K}$, the corresponding inequality $\boldsymbol{\alpha}^T\boldsymbol{d} \leq 0$ is valid for $\Delta^{int}_{\boldsymbol{w}}$ and violated by $\boldsymbol{d}^\star$ ($\boldsymbol{\alpha}^T\boldsymbol{d}^\star > 0$). From

this set of cutting planes, we compute the modification of the normalization weights as

$$\tilde{\boldsymbol{w}} = \frac{\theta}{|\mathcal{K}|} \sum_{\boldsymbol{\alpha} \in \mathcal{K}} \boldsymbol{\alpha}. \tag{6.5}$$

The perturbation is scaled according to the number of cutting planes added, and it is multiplied by a coefficient θ, the value of which is updated as for Dir (Section 6.4.3). The theoretical framework for this strategy is given in Proposition 35, and a geometric interpretation is given in Figure 6.3.

Proposition 35. *Assume that the current solution $\boldsymbol{x}^k$ is not optimal for* SPP. *Let $\boldsymbol{d}^\star \in Ext\,(\Delta_{\boldsymbol{w}})$ be a non-column-disjoint optimal solution of* Mna$^{\boldsymbol{w}}$ *and $\boldsymbol{d^0} \in \Delta^{int}_{\boldsymbol{w}}$ an integral augmenting direction. Let $\boldsymbol{\alpha} \in \mathbb{R}^n$ be the coefficient vector of a cut $\boldsymbol{\alpha}^T\boldsymbol{x} \leq 0$ that separates $\boldsymbol{d}^\star$ from $\Delta^{int}_{\boldsymbol{w}}$, and $\theta > 0$. Consider the perturbation of the normalization weights $\tilde{\boldsymbol{w}} = \theta\boldsymbol{\alpha}$, and let the updated vector be $\boldsymbol{w'} := \boldsymbol{w} + \tilde{\boldsymbol{w}}$. Let $\boldsymbol{d}^{\star\prime} = \boldsymbol{d}^\star / \left(\boldsymbol{w'}^T\boldsymbol{d}^\star\right)$ and $\boldsymbol{d^{0\prime}} = \boldsymbol{d^0} / \left(\boldsymbol{w'}^T\boldsymbol{d^0}\right)$ be the scalings of $\boldsymbol{d}^{\star\prime}$ and $\boldsymbol{d^{0\prime}}$ for the new section of the cone of feasible directions. Then,*

(i) the cost of the direction associated with $\boldsymbol{d}^\star$ worsens after the update of the normalization weights, i.e.,

$$\boldsymbol{c}^T\boldsymbol{d}^{\star\prime} > \boldsymbol{c}^T\boldsymbol{d}^\star; \tag{6.6}$$

(ii) the cost of the direction associated with $\boldsymbol{d^0}$ improves after the update of the normalization weights, i.e.,

$$\boldsymbol{c}^T\boldsymbol{d^{0\prime}} \leq \boldsymbol{c}^T\boldsymbol{d^0}. \tag{6.7}$$

Proof. Let $\boldsymbol{d}^\star$, $\boldsymbol{d^0}$, $\boldsymbol{\alpha}$, θ, $\tilde{\boldsymbol{w}}$, $\boldsymbol{w'}$, $\boldsymbol{d}^{\star\prime}$, and $\boldsymbol{d^{0\prime}}$ be as defined in the statement of Proposition 35.

(i) The cost of $\boldsymbol{d}^{\star\prime}$ satisfies

$$\boldsymbol{c}^T\boldsymbol{d}^{\star\prime} = \boldsymbol{c}^T\left(\frac{\boldsymbol{d}^\star}{\boldsymbol{w'}^T\boldsymbol{d}^\star}\right) = \frac{\boldsymbol{c}^T\boldsymbol{d}^\star}{\boldsymbol{w}^T\boldsymbol{d}^\star + \tilde{\boldsymbol{w}}^T\boldsymbol{d}^\star} = \frac{\boldsymbol{c}^T\boldsymbol{d}^\star}{1 + \theta\left(\boldsymbol{\alpha}^T\boldsymbol{d}^\star\right)}. \tag{6.8}$$

By definition, $\alpha^T\boldsymbol{d}^\star > 0$ and $\theta > 0$, so $1 + \theta\left(\boldsymbol{\alpha}^T\boldsymbol{d}^\star\right) > 1$. Because $\boldsymbol{x}^k$ is not optimal for SPP, $z^\star_{\text{Mna}^{\boldsymbol{w}}} < 0$ and therefore $\boldsymbol{c}^T\boldsymbol{d}^\star < 0$. Thus, Equation 6.6 holds.

(ii) We will apply the same reasoning to $\boldsymbol{d^0}$:

$$\boldsymbol{c}^T\boldsymbol{d^{0\prime}} = \boldsymbol{c}^T\left(\frac{\boldsymbol{d^0}}{\boldsymbol{w'}^T\boldsymbol{d^0}}\right) = \frac{\boldsymbol{c}^T\boldsymbol{d^0}}{\boldsymbol{w}^T\boldsymbol{d^0} + \tilde{\boldsymbol{w}}^T\boldsymbol{d^0}} = \frac{\boldsymbol{c}^T\boldsymbol{d^0}}{1 + \theta\left(\boldsymbol{\alpha}^T\boldsymbol{d^0}\right)}. \tag{6.9}$$

By definition, $\boldsymbol{\alpha}^T\boldsymbol{d^0} \leq 0$ and $\theta > 0$, so $1 + \theta\left(\boldsymbol{\alpha}^T\boldsymbol{d}^\star\right) \leq 1$. Because $\boldsymbol{d^0}$ is an augmenting direction, $\boldsymbol{c}^T\boldsymbol{d^0} < 0$. Thus, Equation 6.7 holds. □

Recall that when we apply the HYP update, the cuts computed to generate the perturbation $\tilde{\boldsymbol{w}}$ are not added to the formulation of MNA$^{\boldsymbol{w}}$ even if they are technically available. The addition of cutting planes slows the solution process and often pushes $\boldsymbol{d}^\star$ toward a fractional direction that is an extreme point created by the cutting planes. HYP tends to push the optimal solution further, until it reaches an extreme point of the original cone that is more likely to be an integral direction.

Furthermore, this method raises the issue that $\boldsymbol{w'}$ may not guarantee that $\Delta_{\boldsymbol{w'}}$ is a proper section of the cone. As stated by Rosat et al. (2015a), a sufficient condition to guarantee this is $\boldsymbol{w}_{\mathcal{P}} \leq \boldsymbol{0}$ and $\boldsymbol{d}_{\mathcal{Z}} > \boldsymbol{0}$. In all cases, we use DEG for the first attempt, i.e., $\boldsymbol{w} = (\boldsymbol{0}, \boldsymbol{\iota}_{\mathcal{Z}})$. In both primal odd-cycle and clique inequalities, the coefficients of some variables in $\mathcal{P}$ take positive values. Therefore, if $\theta \neq 0$, one of these coefficients w_j ($j \in \mathcal{P}$) strictly increases. Since its original value is 0, it becomes positive and the condition $\boldsymbol{w}_{\mathcal{P}} \leq \boldsymbol{0}$ no longer holds.

This theoretically forbids the increase of every w_j for $j \in \mathcal{P}$ under these sufficient conditions. However, $\tilde{\boldsymbol{w}}$ is a continuous function of θ, and there therefore exists a neighborhood of $\theta = 0$ where $(\boldsymbol{w} + \tilde{\boldsymbol{w}})^T \boldsymbol{d} = 1$ is a bounded section of the cone. Although nothing guarantees that the value chosen for θ belongs to that neighborhood, the case of a direction going to infinity did not occur in any of our numerical experiments. If it were to occur, we would decrease the value of θ until $\Delta_{\boldsymbol{w}}$ is bounded or the maximum number Q of consecutive updates of the normalization constraint is reached.

6.5 Benchmark

The crew pairing problem involves determining a set of pairings (where a pairing is a sequence of flights and layovers that starts and ends at the same base) that covers all the scheduled flights at minimal cost over the planning horizon. For a review of aircrew scheduling, see Kasirzadeh et al. (2015). We concentrate on the crew pairing problem.

6.5.1 Set partitioning and crew pairing problem

Model. The crew pairing problem is usually solved by column generation embedded within a branch-and-bound scheme (see Desaulniers et al. (1997)). In the underlying Dantzig–Wolfe decomposition, most constraints of the master problem are set-partitioning constraints. Each of the rows $i \in \{1, \ldots, m\}$ represents a scheduled flight, and each column $j \in \{1, \ldots, n\}$ corresponds to a legal pairing. The corresponding matrix entry A_{ij} is 1 if flight i is covered by pairing j, and 0 otherwise. A set of pairings covering each scheduled flight exactly once is therefore a binary solution of $\boldsymbol{Ax} = \boldsymbol{e}$, i.e., of SPP. The Dantzig–

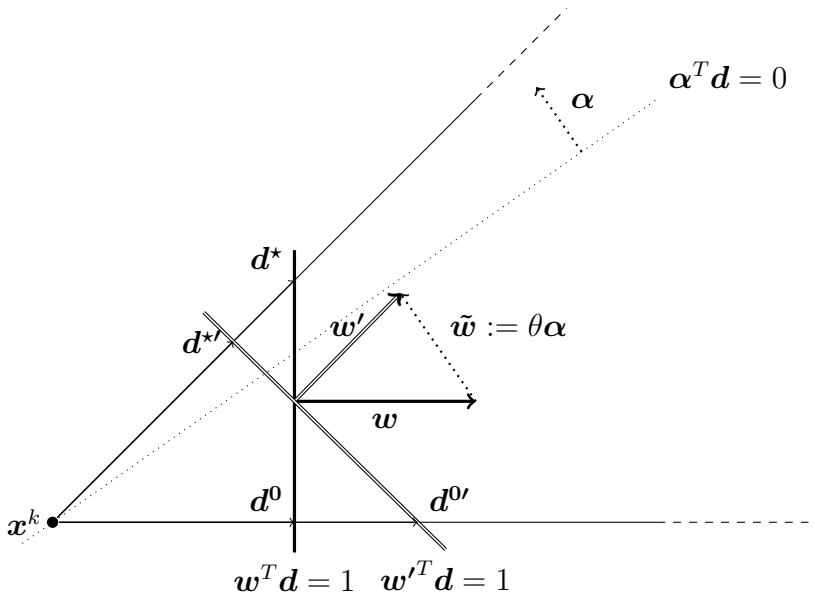


Figure 6.3 Geometric description of HYP. The two plain rays starting from $\boldsymbol{x}^0$ define the cone of feasible directions. Vector $\boldsymbol{w}$ is the original normalization weight vector, and the cone section defined by $\boldsymbol{w}^T\boldsymbol{d} = 1$ is the plain line, orthogonal to $\boldsymbol{w}$. Vectors $\boldsymbol{d}^\star$ and $\boldsymbol{d}^{\mathbf{0}}$ are the normalized rays for vector $\boldsymbol{w}$. Here, we find a cutting plane $\boldsymbol{\alpha}^T\boldsymbol{d} \leq 0$ that separates $\boldsymbol{d}^\star$ from $\Delta_{\boldsymbol{w}}^{int}$ (dotted ray, and corresponding $\boldsymbol{\alpha}$). A penalty $\tilde{\boldsymbol{w}}$ proportional to α ($\theta > 0$) is added to $\boldsymbol{w}$ to define $\boldsymbol{w}'$ (double arrow). The updated normalization constraint $\boldsymbol{w}'^T\boldsymbol{d} = 1$ is the double line, and the associated extreme directions are $\boldsymbol{d}^{\star\prime}$ and $\boldsymbol{d}^{\mathbf{0}\prime}$. In this example, we see that the addition of the perturbation $\tilde{\boldsymbol{w}}$ tends to scale down the cut-off solution ($\boldsymbol{d}^\star \rightarrow \boldsymbol{d}^{\star\prime}$) and scale up the other solution ($\boldsymbol{d}^{\mathbf{0}} \rightarrow \boldsymbol{d}^{\mathbf{0}\prime}$).

Wolfe subproblems generate legal pairings and can be modeled as shortest path problems with resource constraints in time-space networks (Desaulniers et al., 1997). At present, we do not embed the generation of new pairings within ISUD to make it an all-integer column-generation algorithm; this is our long-term goal. For now, we consider set-partitioning instances generated from aircrew scheduling problems by the procedure described below.

Instances ACS**.** The data for the crew pairing problem is taken from the operations of a major North American airline. It consists of five instances (ACS1–5), each describing the flight schedule of a specific type of aircraft. Each instance has three crew bases. A partial schedule for the week prior to the start of the planning horizon is also available. It is composed of partial pairings that start before the beginning of the horizon and must continue during the horizon. We add an extra row to the model for each of these partial pairings, and they are considered equivalent to scheduled flights in the master problem, i.e., each of them must be covered. In the instances presented here, the number of partial pairings never exceeds 3% of the number of flights.

Pairing generation. The pairing problem involves finding a set of pairings that covers all the flights at minimal cost. The solutions must be consistent with the previous week's schedules. Each of the instances is solved using GENCOL [1], which implements column generation embedded in a branch-and-bound framework. To generate legal pairings, we define a subproblem for each base and each day of the planning horizon (more information on the exact structure of the subproblems can be found in the work of Saddoune et al. (2013)). Each column covers one or several flights and may contain at most one partial pairing from the previous week's schedule. We obtain an integer solution by using a variable fixing procedure that selects pairings that must appear in the solution. At each iteration, we fix variables based on their fractional values in the optimal solution of the linear relaxation, and we then perform reoptimization using column generation. The procedure stops when all the variables are integer. All columns generated throughout the variable fixing procedure are recorded, and their union forms the columns of matrix $\boldsymbol{A}$. In the case of duplicate entries, only one occurrence of the column is kept in the matrix. The instances that we use to generate the pairings are those of Kasirzadeh et al. (2015) from which a horizon of nine days is extracted.

1. GENCOL is commercial software developed at the GERAD research center and now owned by AD OPT, a division of KRONOS.

Previous instances. To the ACS instances, we add two aircrew scheduling set-partitioning instances from the OR-Library [2], SPPAA01 and SPPAA04. These two instances were included in the test sets used to evaluate former versions of ISUD (see Rosat et al. (2015a,b); Zaghrouti et al. (2014)).

Characteristics of the instances (Table 6.1). The main characteristics of the instances are given in Table 6.1. The columns m and n respectively give the number of rows and columns of matrix $\boldsymbol{A}$. The number of rows is the number of scheduled flights plus the number of partial pairings. In the benchmark, m ranges from 423 to 1,706 and n from 7,195 to 115,940. The GENCOL UB is the best solution value found by the GENCOL branch-and-bound + column-generation procedure. The GENCOL+ISUD UB is the best solution value found by running the default version of ISUD (without the normalization update) using the best solution found by GENCOL as the initial solution. Under the CPLEX heading we give the best upper bound (UB), the best lower bound (LB), the gap between UB and LB (GAP), and the total execution time (TIME) for the instance composed of the columns generated by GENCOL with the default settings of IBM CPLEX [3]. The best UBs found are highlighted in bold.

Table 6.1 Sizes and solutions of the instances.

			GENCOL	GENCOL +ISUD	CPLEX			
Instance	m	n	UB	UB	UB	LB	GAP	TIME (s)
ACS1	454	12,091	80,034	80,033	**80,010**	80,010.0	0.00%	1.1
ACS2	550	20,269	99,506	**99,450**	**99,450**	99,440.1	0.02%	21.7
ACS3	1,624	103,597	227,116	**226,791**	229,800	225,652.1	1.81%	1,800.0
ACS4	1,706	83,114	336,911	**336,891**	341,530	336,859.6	1.37%	1,800.0
ACS5	1,703	115,940	305,530	**305,422**	309,910	304,326.2	1.80%	1,800.0
SPPAA01	423	7,195	.	.	**56,137**	56,132.8	0.01%	23.8
SPPAA04	803	8,904	.	.	**26,374**	26,374.0	0.00%	13.8

6.5.2 Initial solutions, primal information, and full benchmark

To test an integral simplex algorithm, we also need an initial solution.

Primal Information. We define the *primal information* of a solution to be the percentage of consecutive flights in that solution that are also consecutive in the optimal schedule.

2. The OR-Library is a collection of test data sets for a variety of operations research problems. It is maintained by John E. Beasley and available at `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`.

3. Version 12.6.0 of CPLEX was run with the default parameters and a time limit of 1,800 s on a Linux PC with eight processors of 3.4 GHz.

In practice, this quantity is not known precisely a priori, but the following two observations hold for industrial aircrew scheduling problems. First, crews do not often change planes during their rotations, and the pairings therefore have many consecutive tasks in common with those of the aircraft routes. Second, when the schedule is updated, e.g., because of unforeseen events, the reoptimized schedule usually has many pieces in common with the original schedule (airlines generally add penalties in the objective function to discourage changes). Thus, many consecutive tasks from the initial paths (aircraft routes or the original schedule) remain consecutive in the optimal schedule. In aircrew scheduling, the figure is generally around 75% for the advance planning problem (Zaghrouti et al., 2014) and significantly higher for reoptimization.

Initial Solutions. For the preceding reasons, we perturb the best-known solutions to generate initial solutions that contain a similar level of primal information to that arising in practice when aircraft routes are used as initial solutions. These solutions are generally infeasible, so we assign the perturbed columns a high cost, as is done in practice for schedules that must follow aircraft routes. We use the perturbation method of Zaghrouti et al. (2014), based on crossovers, unions, and splittings of columns. The input parameter π of that method is the proportion of columns of the optimal solution that appear in the generated initial solution. For all the instances, we created initial solutions with π ranging from 0.1 to 0.8 and retained those for which the primal information (given in Table 6.2) was consistent with the typical values. For each value of π retained and each instance, we generated a set of 10 such initial solutions. We retained three levels of perturbation for each instance: *hard*, *moderate*, and *easy* (ordered by increasing primal information). For example, for SPPAA01, the terms hard, moderate, and easy respectively refer to the initial solutions generated for $\pi = 0.1$, $\pi = 0.15$, and $\pi = 0.2$; hence they correspond to 70.4%, 74.4%, and 78.7% of primal information. The values are indicated in Table 6.2, but in Section 6.6 we use the terms hard, moderate, and easy instead of the numerical values. Note that when we reoptimize existing schedules, the primal information is significantly higher because we can use the original schedule as an initial solution, and modifications are usually penalized. As part of a reoptimization process, all the instances presented here would be hard.

6.6 Numerical results

Our numerical results indicate the influence of the type of update (NONE, ALT, RAND, DIR, or HYP), the penalization amplitude θ, and the maximum number of consecutive normalization updates Q. The tests were performed on a Linux PC with eight processors of 3.4 GHz.

Table 6.2 Percentage of primal information in the initial solutions of the benchmark. For each instance and each value of π, the given figure is the arithmetic mean of the primal information of the 10 initial solutions generated. No value is given when the figures are inconsistent with realistic values.

	Input parameter π									
	0.1	0.15	0.2	0.25	0.35	0.4	0.45	0.5	0.55	0.6
ACS1	.	.	.	.	.	.	72.7%	74.5%	77.3%	.
ACS2	.	.	.	.	.	71.4%	73.0%	75.9%	.	.
ACS3	.	.	.	.	.	.	72.1%	74.8%	80.9%	.
ACS4	.	.	.	.	.	.	.	71.9%	76.3%	78.4%
ACS5	.	.	.	.	.	.	70.1%	75.6%	79.1%	.
SPPAA01	70.4%	74.4%	78.7%	.	.	.	.	.	.	
SPPAA04	.	.	69.7%	73.6%	78.0%	.	.	.	.	

6.6.1 Performance of the algorithm

Table 6.3 gives the general results. For each of the instances (INST), we report results for all the strategies (STRAT) for a single set of parameters θ and Q. The values of θ and Q are discussed in the next section. For each level of difficulty (easy, moderate, or hard), the first three columns give the quality of the solution, showing how many instances were solved with a gap between 0 and 1% ($\leq 1\%$), between 1 and 2% ($\leq 2\%$), or $> 2\%$. These thresholds come from industrial observations: a solution within 1% of the best lower bound is considered excellent, and one within 2% is acceptable. We will therefore say that an instance is *unsolved* if the final gap is over 2%. Note that the artificial columns that served to generate the initial solutions were always successfully eliminated. The mean running time is given in column t_{ISUD}. In the last part of Table 6.3, we aggregate the results over the instances for each level of difficulty. The best strategy appears to be HYP because it leaves only 29/210 instances unsolved, whereas the figure is 65 for the version of the algorithm without updates of the normalization constraint. RAND is in second place but has inconsistent performance: it leaves 7/10 easy instances unsolved for instance ACS5, whereas the figure is 6 for ISUD without updates and 1 for HYP. The version of the algorithm without updates, NONE, is obviously faster than the versions with updates. Interestingly, HYP is one of the two fastest strategies with ALT; however, in ALT, only 3 updates are performed (i.e., MNA$^{\boldsymbol{w}}$ is solved at most 4 times before variables are fixed, whereas it is solved at most 8 times in the case of HYP). Finally, the performance of our method is comparable to that of CPLEX, and it tends to be better on large instances (smaller gap and running time).

Table 6.4 presents the nature of the directions found by the various strategies. The figures are aggregated over all the instances available, and the parameters are the same as those in Table 6.3. Column K gives the aggregated number of augmentations performed, and

Table 6.3 Solutions for different update strategies for instances SPPAA01, SPPAA04, ACS1, ACS2, ACS3, ACS4, ACS5.

				Easy				Moderate				Hard			
Inst	Strat	θ	Q	$\leq 1\%$	$\leq 2\%$	$> 2\%$	t_{ISUD}	$\leq 1\%$	$\leq 2\%$	$> 2\%$	t_{ISUD}	$\leq 1\%$	$\leq 2\%$	$> 2\%$	t_{ISUD}
	None	.	.	10	0	0	13.59	10	0	0	15.45	9	0	1	16.76
	Alt	.	3	10	0	0	15.33	10	0	0	17.59	9	0	1	24.16
sppaa01	Rand	10	7	10	0	0	27.64	10	0	0	27.73	10	0	0	42.41
	Dir	1	7	10	0	0	20.83	10	0	0	22.77	10	0	0	28.10
	Hyp	0.5	7	10	0	0	15.61	10	0	0	18.40	10	0	0	21.57
	None	.	.	10	0	0	4.98	5	0	5	6.29	6	1	3	7.03
	Alt	.	3	10	0	0	5.63	9	0	1	8.34	6	1	3	9.46
sppaa04	Rand	10	7	10	0	0	8.39	7	0	3	18.30	7	1	2	17.09
	Dir	1	7	10	0	0	6.29	8	0	2	12.09	8	0	2	11.29
	Hyp	0.5	7	10	0	0	6.36	7	1	2	9.75	7	0	3	11.55
	None	.	.	8	2	0	22.57	10	0	0	22.44	9	0	1	25.53
	Alt	.	3	10	0	0	23.95	10	0	0	22.88	9	0	1	53.27
acs1	Rand	10	7	10	0	0	26.78	10	0	0	23.24	10	0	0	36.14
	Dir	1	7	8	0	2	31.78	10	0	0	21.66	10	0	0	29.85
	Hyp	0.5	7	10	0	0	26.06	10	0	0	24.47	10	0	0	29.76
	None	.	.	10	0	0	35.22	8	0	2	43.95	7	0	3	53.44
	Alt	.	3	10	0	0	37.94	10	0	0	47.85	10	0	0	57.27
acs2	Rand	10	7	9	1	0	47.06	10	0	0	52.43	10	0	0	89.24
	Dir	1	7	10	0	0	42.63	10	0	0	49.80	10	0	0	66.95
	Hyp	0.5	7	10	0	0	38.63	9	1	0	48.46	10	0	0	52.90
	None	.	.	1	3	6	906.83	1	5	4	885.11	0	3	7	1236.32
	Alt	.	3	2	5	3	1167.84	1	6	3	1198.52	0	2	8	1607.18
acs3	Rand	10	7	6	4	0	1563.13	5	2	3	1608.92	1	4	5	1727.99
	Dir	1	7	2	4	4	1380.35	3	6	1	1238.15	1	2	7	1659.56
	Hyp	0.5	7	5	4	1	1196.71	4	4	2	1193.12	1	4	5	1380.25
	None	.	.	6	2	2	756.75	4	3	3	874.88	4	2	4	992.49
	Alt	.	3	7	3	0	1023.02	5	5	0	1100.34	4	3	3	1331.61
acs4	Rand	10	7	9	1	0	1402.22	8	2	0	1528.25	5	2	3	1691.95
	Dir	1	7	6	3	1	1138.15	6	4	0	1356.47	3	4	3	1592.75
	Hyp	0.5	7	8	2	0	967.14	7	2	1	1135.09	4	4	2	1293.43
	None	.	.	2	2	6	1027.76	0	1	9	1109.81	1	0	9	1431.47
	Alt	.	3	6	3	1	1233.70	0	1	9	1522.62	0	0	10	1731.57
acs5	Rand	10	7	1	2	7	1707.39	3	2	5	1774.32	1	0	9	1776.19
	Dir	1	7	2	4	4	1544.01	0	4	6	1723.64	0	2	8	1799.18
	Hyp	0.5	7	8	1	1	1289.59	2	4	4	1588.29	0	2	8	1639.56
	None	.	.	47	9	14	.	38	9	23	.	36	6	28	.
	Alt	.	3	55	11	4	.	45	12	13	.	38	6	26	.
Total	Rand	10	7	55	8	7	.	53	6	11	.	44	7	19	.
	Dir	1	7	48	11	11	.	47	14	9	.	42	8	20	.
	Hyp	0.5	7	61	7	2	.	49	12	9	.	42	10	18	.

column #Mna gives the total number of augmentation problems Mna solved throughout all the executions. We see that updating the normalization constraint significantly increases the number of augmentation problems solved, with a much smaller increase in the number of augmentations. This can be explained as follows: when the augmentation problem fails to return an integral solution, the static version performs fixing. In the version with updates, the normalization constraint may be updated 7 times before we fix the same set of variables. Therefore, for the same sequence of solutions, the versions with updates may solve up to 8 times more Mna problems than the static version. Furthermore, they perform more augmentation steps, and so yield better solutions.

Columns #frac and #int give the proportion of fractional and integral directions. Obviously, the directions tend to be less fractional for None because the algorithm stops more quickly when it gets into difficulty. The strategies for which 8 augmentation problems may

be solved before fixing occurs have similar ratios, while the ALT strategy (only up to 4 augmentation problems solved before fixing occurs) is in between. Thus, updating the normalization weights is not likely to immediately produce integral directions, but success may be expected after a few modifications. Finally, the columns #ORIG, #FIX, and #UPD give the proportion of integral directions found without variable fixing or updates, after variable fixing, and after an update. There are no major differences between the five strategies in terms of the augmentations obtained without fixing/updating. However, when updates are performed, they are usually the decisive actions in the search for integral directions, i.e., integrality usually comes from these modifications. Augmentations tend to be better when they are used with updates because fixing variables reduces the domain of MNA and thus the space where the algorithm looks for augmentations. All in all, better directions are to be expected from a larger domain than from a restricted one.

From the aforementioned results, we conclude that the four strategies of dynamic adjustment clearly dominate the NONE strategy. The HYP update produces the best results since it lowers the proportion of unsolved problems to 14% whereas that figure is 31% for NONE. The increase of the computation time due to the update remains small compared to the static version. For large problems (ACS3–ACS5), CPLEX is unable to produce solutions as good as those that we produce within the same time limit.

Table 6.4 Augmenting directions found by the algorithm.

STRAT	K	#MNA	#FRAC	#INT	#ORIG	#FIX	#UPD
NONE	29784	55678	0.47	0.53	0.77	0.23	0.00
ALT	31361	100128	0.69	0.31	0.74	0.03	0.24
RAND	31807	120698	0.74	0.26	0.79	0.01	0.20
DIR	30805	133386	0.77	0.23	0.72	0.01	0.28
HYP	31334	119414	0.74	0.26	0.78	0.01	0.20

6.6.2 Influence of the parameters

In this section, we study the influence of the parameters, i.e., of the maximum number of consecutive updates of the normalization constraint Q and the perturbation weight θ. We also give insight into the influence of the time limit. We study the influence of the parameters on the ACS3 instances, because it is sufficiently difficult and accurately portrays the overall results.

Table 6.5 gives results for the influence of Q. The results are shown for a random update (RAND), but we observed similar behavior for the other strategies. We chose the values of Q so that the number of MNA^w problems solved before variable fixing is a power of 2 ($Q = 1$,

3, 7, etc.). The time limits are almost never reached except for $Q = 31$ and $t_{\max} = 3,600$. A comparison of the last two rows of the table shows what typically happens in practice: provided the time limit is not too low, no difference is observed between the solutions. In the cases reported in Table 6.5, for each strategy and each level of perturbation, at most three instances reach the time limit. Furthermore, of these three cases, at most one provides a solution with a gap over 2%. Choosing $Q = 7$ is a good compromise between computational time and solution quality: for that value, only 8 instances out of 30 remain unsolved with a time limit of $1,800$ s. Hence, we chose that value to test our algorithm on the benchmark.

Table 6.5 Influence of parameter q on performance of algorithm. The tests are performed on instance ACS3 with update strategy RAND and $\theta = 10$.

		Easy			Moderate			Hard		
Q	$t_{\max}$	$\leq 1\%$	$\leq 2\%$	$> 2\%$	$\leq 1\%$	$\leq 2\%$	$> 2\%$	$\leq 1\%$	$\leq 2\%$	$> 2\%$
1	1800	3	4	3	3	2	5	1	3	6
3	1800	4	2	4	3	4	3	2	2	6
7	1800	6	4	0	5	2	3	1	4	5
15	3600	9	1	0	4	4	2	1	2	7
31	3600	5	3	1	5	4	1	3	2	5
31	10800	5	3	1	5	4	1	3	3	4

Table 6.6 gives results for the influence of θ for the update strategies DIR and HYP. For these instances, the time limit was never reached. The column STRAT gives the update strategy. The results show that for DIR, $\theta = 1$ is the best value, and for HYP, $\theta = 0.5$ yields the best results. Therefore, we used these values for the global results given in Tables 6.3 and 6.4.

Table 6.6 Influence of parameter θ on performance of algorithm. The tests are performed on instance ACS3 with $Q = 7$.

		Easy				Moderate				Hard			
STRAT	θ	$\leq 1\%$	$\leq 2\%$	$> 2\%$	t_{ISUD}	$\leq 1\%$	$\leq 2\%$	$> 2\%$	t_{ISUD}	$\leq 1\%$	$\leq 2\%$	$> 2\%$	t_{ISUD}
DIR	0.5	2	5	3	1304.74	3	5	2	1262.85	1	1	8	1627.37
	1.0	2	4	4	1374.08	3	6	1	1253.76	1	2	7	1626.40
	5.0	3	3	4	1340.40	4	4	2	1443.31	1	1	8	1673.75
	10.0	3	3	4	1346.61	4	4	2	1439.09	1	3	6	1653.87
HYP	0.5	5	4	1	1188.89	4	4	2	1200.04	1	4	5	1418.29
	1.0	4	5	1	1165.08	5	3	2	1198.11	1	4	5	1388.62
	5.0	5	2	3	1059.82	4	2	4	1156.45	1	6	3	1356.43
	10.0	4	3	3	1130.40	3	4	3	1164.54	0	4	6	1400.72

6.7 Conclusions

We have proposed a new variant of the ISUD algorithm. We modified ISUD so that the normalization of the augmentation problem is dynamically updated whenever the direction leads to a fractional direction. We have proposed four strategies to update this constraint so as to penalize fractional directions. Two of the updates (DIR and HYP) are based on theoretical observations, and the other two are based on experimental observations. We have

shown that our version of the algorithm yields better results than the former version and than classical branch-and-bound techniques on a benchmark of industrial aircrew scheduling instances. The benchmark we have proposed is, to the best of our knowledge, comparable to no other from the literature. It provides large-scale instances with up to 1,700 flights and 115,000 pairings, and the instances are given in a set-partitioning form together with initial solutions that accurately mimic those of real-world applications. Our strategies allow us to find better solutions than those found by the previous version of the algorithm and to find better solutions than CPLEX finds in the same time. Our work shows the strong potential of primal algorithms for solving the crew scheduling problem, a key challenge for large airlines both financially significant and notably hard to solve.

Acknowledgements

This work was supported by a RD COOP research grant from CRSNG and Kronos Inc. (RD-CPJ 477127-14).

CHAPITRE 7 DISCUSSION GÉNÉRALE

Dans cette thèse, nous avons présenté plusieurs méthodes favorisant l'intégralité de l'amélioration dans l'algorithme du simplexe en nombres entiers. Nous avons montré que ce dernier peut concurrencer les méthodes traditionelles de résolution de problèmes de planification industriels, notament en transport aérien. Nos méthodes se basent sur des reformulations tant statiques que dynamiques du problème d'augmentation (chapitres 4 et 6) et sur l'amélioration de sa relaxation linéaire grâce à des méthodes de plans coupants primaux (chapitre 5).

7.1 Synthèse des travaux

Au chapitre 4, nous avons généralisé la formulation du problème d'ISUD en particulier l'expression de sa contrainte de normalisation. Nous avons montré que le choix de la section du cône des directions réalisables sur laquelle le problème est défini influence grandement celui de la direction d'amélioration, et donc la probabilité que celle-ci mène vers une nouvelle solution entière. Nous avons aussi adapté la théorie préexistante liée à ISUD afin qu'elle s'applique au cas d'une contrainte de normalisation générique et avons démontré de nouvelles propriétés de l'algorithme. Nous avons fait des recommandations quant au choix des coefficients de la contrainte de normalisation afin de pénaliser les directions fractionnaires au profit des directions entières. Les résultats numériques que nous avons présentés démontrent le fort potentiel de notre approche. Alors que la version originale d'ISUD permet de résoudre 78% des instances de transport aérien du benchmark considéré, 100% sont résolues grâce à l'un, au moins, des modèles que nous proposons.

Au chapitre 5, nous avons montré qu'il est possible d'adapter des méthodes de plans coupants provenant du domaine de la programmation linéaire en nombres entiers au cas de l'algorithme ISUD. Nous avons prouvé que des coupes peuvent être transférées dans le problème d'augmentation pour en améliorer la relaxation linéaire et nous avons caractérisé l'ensemble de ces coupes comme l'ensemble non vide des coupes primales saturées par la solution courante. Le caractère non vide de l'ensemble des coupes transférables est un des résultats les plus importants de ce chapitre car il valide la cohérence de notre approche. Nous avons proposé des algorithmes efficaces pour la détermination de coupes primales de cycles impairs et de cliques. Les résultats numériques présentés soulignent la pertinence de notre approche et l'ajout de coupes primales au problème d'augmentation a permis d'améliorer la qualité des solutions trouvées par ISUD et de prouver l'optimalité de certaines solutions sur un grand sous-ensemble du domaine réalisable.

Au chapitre 6, nous avons étendu les techniques exposées au chapitre 4 et développé des méthodes de modification dynamique des coefficients de la contrainte de normalisation du problème d'augmentation. Nous avons conçu plusieurs stratégies de mise-à-jour de ces coefficients basées sur des observations théoriques et pratiques. La première de ces stratégies reprend directement les travaux exposés au chapitre 4, la seconde vise à pénaliser directement la direction fractionnaire déterminée par l'algorithme, et la troisième tire parti des résultats du chapitre 5 en modifiant les coefficients de la contrainte de normalisation grâce à ceux de coupes primales. Cette version de l'algorithme a été testée sur un nouvel ensemble d'instances provenant de l'industrie du transport aérien comparable à aucun autre, à notre connaissance. Il s'agit en effet de grands problèmes d'horaires de personnel navigant allant jusqu'à 1 700 vols et 115 000 rotations, donc autant de contraintes et de variables. Ils sont disponibles sous forme de problèmes de partitionnement pour lesquels nous fournissons des solutions initiales semblables à celles dont on disposerait en milieu industriel. Les résultats numériques obtenus montrent que l'algorithme est compétitif avec les méthodes conventionnelles basées sur le principe de séparation-et-évaluation.

7.2 Limites des solutions proposées et améliorations futures

Avant d'indiquer les limites de notre approche, les améliorations qu'il serait bon de lui apporter ainsi que nos recommandations de mise en place, il convient de souligner la différence entre les limites individuelles des méthodes que nous avons proposées et celles de l'approche globale liée à ISUD. Les limites propres à chacune des méthodes ont déjà été débattues dans les chapitres qui leurs sont consacrés (chapitres 4 à 6) et nous ne revenons pas sur elles ici. Nous analysons toutefois les deux limites principales de l'approche globale du simplexe en nombres entiers telle qu'elle est développée dans cette thèse.

D'une part, sous sa forme actuelle, ISUD ne s'applique qu'au problème de partitionnement pur et ne peut résoudre un modèle incluant d'autres types de contraintes. Or, pour être compétitif face aux méthodes conventionnelles, ISUD doit permettre de résoudre une gamme de problèmes plus large. Il ne s'agit pas de s'attaquer au cadre général de la programmation linéaire en nombres entiers, mais plutôt à une famille de modèles récurrents en planification que sont les problèmes de type "partitionnement avec contraintes supplémenaires". Ces problèmes sont majoritairement composés de contraintes de type partitionnement, auxquelles s'ajoutent un nombre restreint de contraintes de nature différente (convention collective, contraintes d'effectifs, etc.). Afin de développer un algorithme pour cette famille de problèmes nous faisons les recommandations suivantes : étant donnée une solution réalisable, les contraintes saturées par cette solution devraient être traitées de la même façon que le sont

les coupes primales dans notre algorithme du chapitre 5, c'est-à-dire transférées dans le problème d'augmentation. Les contraintes non saturées pourraient être prises en compte dans l'objectif du problème d'augmentation de façon similaire à une relaxation lagrangienne. Les travaux réalisés dans cette thèse sur les coupes primales et la contrainte de normalisation s'adapteraient facilement à un tel cadre algorithmique.

D'autre part, le futur d'ISUD ne pourra être à la hauteur des attentes placées en lui tant que des techniques de génération de colonnes ne lui auront été intégrées. L'expérience a montré que l'intégration de la génération de colonnes au schéma de séparation-et-évaluation améliore fortement la qualité des solutions obtenues sur les problèmes rencontrés en transport aérien. Ainsi, tant que le schéma d'utilisation du simplexe en nombres entiers consistera en une étape de génération d'un ensemble de rotations suivie d'une étape de résolution sans génération de nouvelles rotations, il ne sera vraissemblablement pas implanté en milieu industriel. Ajouter des colonnes en cours de résolution permet en effet de réduire le saut d'intégralité – *integrality gap* – qui est élevé dans le cas d'une génération a priori ; la limite n'est donc pas algorithmique, mais bien théorique. Dans le schéma de séparation-et-évaluation, les bornes inférieures sont obtenues par la résolution de relaxations linéaires et les bornes supérieures par le branchement (cas d'une minimisation). Afin de développer la génération de colonnes en nombres entiers – *all-integer column generation* – nous préconisons de calculer les bornes inférieures grâce à la borne de Dantzig-Wolfe par exemple, alors que la borne supérieure décroîtra naturellement tandis que l'algorithme améliorera la solution (entière). La machinerie de génération des colonnes devra donc servir à l'amélioration des bornes inférieure et supérieure alors qu'elle ne s'attache qu'à celle de la borne inférieure dans le cas de la séparation-et-évaluation. Le nouvel algorithme requerra donc de repenser l'étape de génération et de l'adapter pour améliorer alternativement l'une et l'autre des deux bornes. La caractéristique originale des méthodes primales, c'est-à-dire le fait que toutes les solutions obtenues lors du processus sont entières, représente donc à la fois un de ses plus grands avantages et une pierre d'achoppement sur le chemin vers ce nouvel algorithme. C'est un avantage car des solutions réalisables seront rapidement déterminées puis améliorées, mais aussi un inconvénient car il faudra repenser le processus de génération dans l'objectif d'une amélioration des deux types de bornes. Nous recommandons donc de s'inspirer (1) des procédures de génération développées depuis la démocratisation de la génération de colonnes dans les années 1990 et de celui concernant la borne de Dantzig-Wolfe en vue d'améliorer la borne inférieure et (2) du travail contemporain de Rönnberg (2012) concernant la conception d'un processus de génération de colonnes spécifique au schéma d'un algorithme primal.

CHAPITRE 8 CONCLUSION

En conclusion, cette thèse a étendu la théorie existante concernant l'algorithme du simplexe en nombres entiers ISUD, nous l'avons rapproché de la famille des algorithmes primaux, avons proposé une reformulation du problème d'augmentation, une méthode de plans coupants améliorant sa relaxation linéaire ainsi qu'un algorithme de mise-à-jour de sa contrainte de normalisation. Les méthodes que nous avons développées ont ainsi permis d'augmenter le taux de directions entières trouvées par l'algorithme. L'application à plusieurs problèmes de la litérature ainsi qu'à un ensemble de nouvelles instances provenant de l'industrie du transport aérien a montré le potentiel qu'ont les algorithmes primaux pour résoudre des problèmes de planification de personnel navigant, problèmes clés pour les compagnies aériennes, tant par leur complexité intrinsèque que par leurs conséquences économiques et financières.

RÉFÉRENCES

Air Canada, "Annual report 2013", 2014.

E. Balas & M. W. Padberg, "On the set-covering problem : 2 - an algorithm for set partitioning", *Operations Research*, vol. 23, no. 1, pp. 74–90, 1975. DOI : 10.1287/opre.23.1.74

——, "Set partitioning : A survey", *SIAM Review*, vol. 18, no. 4, pp. 710–760, 1976. DOI : 10.1137/1018115

R. Baldacci & A. Mingozzi, "A unified exact method for solving different classes of vehicle routing problems", *Mathematical Programming*, vol. 120, pp. 347–380, 2009.

A. Ben-Israel & A. Charnes, "On some problems of diophantine programming", *Cahiers du Centre d'Études de Recherche Opérationnelle*, vol. 4, pp. 215–280, 1962.

M. Benichou, J. M. Gauthier, G. Hentges, & G. Ribiere, "The efficient solution of large-scale linear programming problems : Some algorithmic techniques and computational results", *Mathematical Programming*, vol. 13, no. 1, pp. 280–322, 1977.

F. Bonnans & S. Gaubert, *Recherche opérationnelle : aspects mathématiques et applications.* École polytechnique, 2010.

H. Bouarab, I. Elhallaoui, A. Metrane, & F. Soumis, "Dynamic constraint and variable aggregation in column generation", HEC Montréal, Canada, Les Cahiers du GERAD G-2014-82, 2014.

R. J. Dakin, "A tree-search algorithm for mixed integer programming problems", *The Computer Journal*, vol. 8, no. 3, pp. 250–255, 1965. DOI : 10.1093/comjnl/8.3.250

G. B. Dantzig & P. Wolfe, "Decomposition principle for linear programs", *Operations Research*, vol. 8, no. 1, pp. 101–111, 1960.

G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M. M. Solomon, & F. Soumis, "Crew pairing at Air France", *European Journal of Operational Research*, vol. 97, no. 2, pp. 245–259, 1997.

J. Desrosiers, Y. Dumas, M. M. Solomon, & F. Soumis, "Time constrained routing and scheduling", *Handbooks in operations research and management science*, vol. 8, pp. 35–139, 1995.

J. Edmonds, "Maximum matching and a polyhedron with 0,1-vertices", *Journal of Research of the National Bureau of Standards B*, vol. 69, no. 1965, pp. 125–130, 1965.

F. Eisenbrand, G. Rinaldi, & P. Ventura, "Primal separation for 0/1 polytopes", *Mathematical Programming*, vol. 95, no. 3, pp. 475–491, 2003. DOI : 10.1007/s10107-002-0309-y

I. Elhallaoui, D. Villeneuve, F. Soumis, & G. Desaulniers, "Dynamic aggregation of set-partitioning constraints in column generation", *Operations Research*, pp. 632–645, 2005.

I. Elhallaoui, A. Metrane, F. Soumis, & G. Desaulniers, "Multi-phase dynamic constraint aggregation for set partitioning type problems", *Mathematical Programming*, vol. 123, pp. 345–370, 2010.

I. Elhallaoui, A. Metrane, G. Desaulniers, & F. Soumis, "An improved primal simplex algorithm for degenerate linear programs", *INFORMS Journal on Computing*, vol. 23, no. 4, pp. 569–577, 2011.

R. T. Firla, B. Spille, & R. Weismantel, *A primal analogue of cutting plane algorithms.* Citeseer, 1999.

R. T. Firla, U.-U. Haus, M. Köppe, B. Spille, & R. Weismantel, "Integer pivoting revisited", 2001.

R. S. Garfinkel & G. L. Nemhauser, "The set-partitioning problem : Set covering with equality constraints", *Operations Research*, vol. 17, no. 5, pp. 848–856, 1969.

J.-B. Gauthier, J. Desrosiers, & M. Lübbecke, "Vector space decomposition for linear programs", HEC Montréal, Canada, Les Cahiers du GERAD G-2015-26, 2015.

P. E. Gill, W. Murray, M. A. Saunders, & M. H. Wright, "A practical anti-cycling procedure for linearly constrained optimization", *Mathematical Programming*, vol. 45, no. 1-3, pp. 437–474, 1989. DOI : 10.1007/BF01589114

F. Glover, "A new foundation for a simplified primal integer programming algorithm", *Operations Research*, vol. 16, pp. 727–740, 1968.

A. V. Goldberg & R. E. Tarjan, "Finding minimum-cost circulations by canceling negative cycles", *Journal of the ACM*, vol. 36, no. 4, pp. 873–886, 1989.

R. E. Gomory, "Outline of an algorithm for integer solutions to linear program", *Bulletin of the American Mathematical Society*, vol. 64, no. 5, pp. 275–278, 1958.

——, “All-integer integer programming algorithm”, *Industrial scheduling*, pp. 193–206, 1963.

H. J. Greenberg, *Design and Implementation of Optimization Software*. Sijthoff & Noordhoff, 1978, ch. Pivot selection tactics, pp. 109–142.

M. Groiez, “Étude et séparation des inégalités valides pour des problèmes de partitionnement et de couverture”, Thèse de doctorat, École Polytechnique de Montréal, 2013.

U.-U. Haus, M. Köppe, & R. Weismantel, “A primal all-integer algorithm based on irreducible solutions”, *Mathematical Programming*, vol. 96, no. 2, pp. 205–246, 2003. DOI : 10.1007/s10107-003-0384-8

K. L. Hoffman & M. Padberg, “Solving airline crew scheduling problems by branch-and-cut”, *Management Science*, vol. 39, no. 6, pp. 657–682, 1993. DOI : 10.1287/mnsc.39.6.657

M. J. Kallio & E. L. Porteus, “A class of methods for linear programming”, *Mathematical Programming*, vol. 14, no. 1, pp. 161–169, 1978.

A. Kasirzadeh, M. Saddoune, & F. Soumis, “Airline crew scheduling : Models, algorithms, and data sets”, *EURO Journal on Transportation and Logistics*, pp. 1–27, 2015.

A. N. Letchford & A. Lodi, “Primal cutting plane algorithms revisited”, *Mathematical Methods of Operations Research*, vol. 56, no. 1, pp. 67–81, 2002. DOI : 10.1007/s001860200200

——, “Primal separation algorithms”, *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 3, pp. 209–224, 2003.

——, “An augment-and-branch-and-cut framework for mixed 0-1 programming”, dans *Combinatorial Optimization—Eureka, You Shrink!* Springer, 2003, pp. 119–133.

I. Maros, *Computational techniques of the simplex methods. (International series in operations research and management science 61)*. Boston : Kluwer, 2003.

R. E. Marsten & F. Shepardson, “Exact solution of crew scheduling problems using the set partitioning model : Recent successful applications”, *Networks*, vol. 11, no. 2, pp. 165–177, 1981. DOI : 10.1002/net.3230110208

A. Metrane, F. Soumis, & I. Elhallaoui, “Column generation decomposition with the degenerate constraints in the subproblem”, *European Journal of Operational Research*, vol. 207, no. 1, pp. 37–44, 2010.

J. Omer, S. Rosat, V. Raymond, & F. Soumis, “Improved primal simplex : A more general theoretical framework and an extended experimental analysis”, *INFORMS Journal on Computing*, vol. 27, no. 4, pp. 773–787, 2015.

P. R. J. Östergård, “A new algorithm for the maximum-weight clique problem”, *Nordic Journal of Computing*, vol. 8, no. 4, pp. 424–436, 2001.

M. W. Padberg, “On the facial structure of set packing polyhedra”, *Mathematical Programming*, vol. 5, no. 1, pp. 199–215, 1973. DOI : 10.1007/BF01580121

P.-Q. Pan, “A primal deficient-basis simplex algorithm for linear programming”, *Applied Mathematics and Computation*, vol. 196, pp. 898–912, 2008.

A. F. Perold, “A degeneracy exploiting LU factorization for the simplex method”, *Mathematical Programming*, vol. 19, pp. 239–254, 1980.

T. K. Ralphs & M. V. Galati, “Decomposition and dynamic cut generation in integer linear programming”, *Mathematical Programming*, vol. 106, no. 2, pp. 261–285, 2006.

E. Rönnberg, “Contributions within two topics in integer programming : nurse scheduling and column generation”, Thèse de doctorat, Linköping University, 2012.

E. Rönnberg & T. Larsson, “Column generation in the integral simplex method”, *European Journal of Operational Research*, vol. 192, no. 1, pp. 333–342, 2009.

——, “All-integer column generation for set partitioning : Basic principles and extensions”, *European Journal of Operational Research*, vol. 233, no. 3, pp. 529–538, 2014. DOI : http://dx.doi.org/10.1016/j.ejor.2013.08.036

S. Rosat, I. Elhallaoui, F. Soumis, & A. Lodi, “Integral simplex using decomposition with primal cuts”, dans *Experimental Algorithms*, série Lecture Notes in Computer Science, J. Gudmundsson & J. Katajainen, éds. Springer International Publishing, 2014, vol. 8504, pp. 22–33.

S. Rosat, I. Elhallaoui, F. Soumis, & D. Chakour, “Influence of the normalization constraint on the integral simplex using decomposition”, *Discrete Applied Mathematics (accepted for publication)*, 2015.

S. Rosat, I. Elhallaoui, F. Soumis, & A. Lodi, “Primal cuts in the integral simplex using decomposition”, HEC Montréal, Canada, Les Cahiers du GERAD G-2015-44, 2015.

S. Rosat, F. Quesnel, I. Elhallaoui, & F. Soumis, “Dynamic penalization of fractional directions in the integral simplex using decomposition : Application to aircrew scheduling”, HEC Montréal, Canada, Les Cahiers du GERAD G-2016-01, 2016.

A. Rozenknop, R. Wolfler Calvo, L. Alfandari, D. Chemla, & L. Létocart, “Solving the electricity production planning problem by a column generation based heuristic”, *Journal of Scheduling*, vol. 16, no. 6, pp. 585–604, 2013.

M. Saddoune, G. Desaulniers, I. Elhallaoui, & F. Soumis, “Integrated airline crew scheduling : A bi-dynamic constraint aggregation method using neighborhoods”, *European Journal of Operational Research*, vol. 212, no. 3, pp. 445–454, 2011.

M. Saddoune, G. Desaulniers, & F. Soumis, “Aircrew pairings with possible repetitions of the same flight number”, *Computers and Operations Research*, vol. 40, no. 3, pp. 805–814, 2013.

H. M. Salkin & R. D. Koncal, “Set covering by an all-integer algorithm : Computational experience”, *Journal of the ACM*, vol. 20, no. 2, pp. 189–193, 1973.

A. Saxena, “Set-partitioning via integral simplex method”, *Unpublished manuscript, OR Group, Carnegie-Mellon University, Pittsburgh*, 2003.

——, “Three articles on integral simplex method”, Rapp. tech., 2003.

A. Schrijver, *Theory of linear and integer programming.* John Wiley & Sons, 1998.

A. S. Schulz & R. Weismantel, “The complexity of generic primal algorithms for solving general integer programs”, *Mathematics of Operations Research*, vol. 27, no. 4, pp. 681–692, 2002. DOI : 10.1287/moor.27.4.681.305

A. S. Schulz, R. Weismantel, & G. M. Ziegler, “0/1-integer programming : Optimization and augmentation are equivalent”, vol. 979, pp. 473–483, 1995.

B. Spille & R. Weismantel, “Primal integer programming”, dans *Discrete Optimization*, série Handbooks in Operations Research and Management Science, K. Aardal, G. Nemhauser, & R. Weismantel, éds. Elsevier, 2005, vol. 12, pp. 245–276.

M. F. Stallmann & F. Brglez, “High-contrast algorithm behavior : Observation, conjecture, and experimental design”, dans *ACM-FCRC*. New York : ACM, 2007, 549075.

M. Stojković, F. Soumis, & J. Desrosiers, “The operational airline crew scheduling problem”, *Transportation Science*, vol. 32, no. 3, pp. 232–245, 1998.

C. Sumetphong & S. Tangwongsan, “Modeling broken characters recognition as a set-partitioning problem”, *Pattern Recognition Letters*, vol. 33, no. 16, pp. 2270–2279, 2012.

G. L. Thompson, “An integral simplex algorithm for solving combinatorial optimization problems”, *Computational Optimization and Applications*, vol. 22, no. 3, pp. 351–367, sep 2002.

M. Towhidi, J. Desrosiers, & F. Soumis, “The positive edge criterion within COIN-OR’s CLP”, *Computers and Operations Research*, vol. 49, no. 0, pp. 41–46, 2014.

V. Trubin, “On a method of solution of integer linear programming problems of a special kind”, *Soviet Mathematics Doklady*, vol. 10, pp. 1544–1546, 1969.

R. J. Vanderbei, “Linear programming”, *Foundations and extensions, International Series in Operations Research & Management Science*, vol. 37, 2001.

R. Vetschera, “A general branch-and-bound algorithm for fair division problems”, *Computers & Operations Research*, vol. 37, no. 12, pp. 2121–2130, 2010.

L. A. Wolsey & G. L. Nemhauser, *Integer and Combinatorial Optimization*, série Wiley Series in Discrete Mathematics and Optimization. John Wiley & Sons, 2014.

R. D. Young, “A primal (all-integer) integer programming algorithm”, *Journal of Research of the National Bureau of Standards : B. Mathematics and Mathematical Physics*, pp. 213–250, 1965.

——, “A simplified primal (all-integer) integer programming algorithm”, *Operations Research*, vol. 16, no. 4, pp. 750–782, 1968.

A. Zaghrouti, F. Soumis, & I. El Hallaoui, “Integral simplex using decomposition for the set partitioning problem”, *Operations Research*, vol. 62, no. 2, pp. 435–449, 2014.